Strategien des dynamischen Lastausgleichs für Multi-Domain-Cluster am Beispiel einer parallelen Dendriten-Simulation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Informatiker

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

Fakultät für Mathematik und Informatik

eingereicht von Jakob Erdmann
geb. am 11.07.1980 in Königs Wusterhausen

Betreuer: Dipl-Inf. Christian Kauhaus
Prof. Dr. Dietmar Fey

Jena, 29.05.2007

Kurzfassung

Die Vorgänge bei der Erstarrung von Metallschmelzen sind mitbestimmend für die Qualität industrieller Werkstoffe und werden daher im Rahmen der Materialwissenschaften erforscht. Die Simulation der Strukturen, die sich beim Erstarren bilden, sogenannter Dendriten, ist allerdings sehr rechnenaufwändig.

Um die Rechenleistung der bestehenden Infrastruktur auszunutzen, soll ein paralleles Simulationsprogramm zum Einsatz auf Cluster- und Multi-Cluster-Systemen entwickelt werden. Es zeigt sich allerdings, dass die lokal schwankenden Rechenanforderungen des physikalischen Modells zu ungleicher Lastverteilung zwischen den Rechenknoten führen. Um eine effiziente Parallelisierung zu ermöglichen, wird der Einsatz eines dynamischen Lastausgleichsalgorithmus (DLB) notwendig. Weil das Modell jedoch einer ständigen Weiterentwicklung unterliegt, darf die Effizienz des Lastausgleichs nicht an die Feinjustierung von Algorithmenparametern gebunden sein. Nur so kann verhindert werden, dass jede Modelländerung eine aufwendige Parameterstudie für den DLB-Algorithmus nach sich zieht.

Um diesen Anforderungen zu genügen, wurde ein selbstkalibrierender DLB-Algorithmus entwickelt. Kernstück dieses Algorithmus ist eine Heuristik zur Nutzenabschätzung des Lastausgleichs. Durch die Wahl eines geeigneten Aufteilungsschemas wird der Kommunikationsbedarf sowohl im Einzel- als auch im Multicluster-Betrieb minimiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Kürzel	Beschreibung
BSP	Bulk Synchronous Parallel
CA	Cellular Automaton
DLB	Dynamic Load Balancing
ISO	Immediate Savings Only
MC	Multi Domain Cluster
MPI	Message Passing Interface
PCA	Parallel Cellular Automaton
PLS	Predicted Long-Term Savings
RCB	Recursive Coordinate Bisection
RSS	Remaining Steps Scaling
SFC	Space Filling Curve
WRCB	Weighted Recursive Coordinate Bisection

1 Einleitung

Die vorliegende Arbeit beschreibt die Entwicklung eines parallelen Algorithmus zur Simulation von erstarrenden Metallschmelzen. Das physikalische Modell wird von Materialwissenschaftlern entwickelt, um das Wachstum von dendritischen Strukturen zu erforschen. Dieser Prozess steht im Interesse der Materialforschung, da Dendriten die Belastbarkeit und damit den Wert des untersuchten Werkstoffs entscheidend beeinflussen. Weil die Simulationsgenauigkeit durch den Rechenbedarf des physikalischen Modells beschränkt ist, wird eine parallele Umsetzung eines seriell vorliegenden Programms angestrebt.

Um die bestehenden Infrastrukturen optimal zu nutzen, soll das parallele Programm auch auf Multi-Cluster-Systemen (MC) eingesetzt werden. Die Kommunikationseigenschaften solch einer Rechnerumgebung müssen also berücksichtigt werden. Es zeigt sich, dass die lokal schwankenden Rechenanforderungen des Modells zu ungleicher Lastverteilung zwischen den Rechenknoten führen. Um die Effizienz der Parallelisierung zu ermöglichen, wird der Einsatz eines dynamischen Lastausgleichsalgorithmus (DLB) notwendig. Weil das physikalische Modell jedoch einer ständigen Weiterentwicklung unterliegt, muss ein DLB-Algorithmus gefunden werden, dessen Leistungsfähigkeit nicht an die Feinjustierung von Algorithmenparametern gebunden ist. Nur so kann verhindert werden, dass jede Modelländerung eine aufwendige Parameterstudie zur Kalibrierung des Lastausgleichs nach sich zieht.

Um diesen Anforderungen zu genügen, wurde ein selbstkalibrierender DLB-Algorithmus entwickelt. Die zur Steuerung des Lastausgleichs notwendigen Parameter werden soweit möglich aus Laufzeitmessungen abgeleitet. Um den Einfluss der verbleibenden Parameter zu senken, wird eine Heuristik zur Analyse längerfristiger Lastschwankungen eingesetzt. Durch die Wahl eines geeigneten Aufteilungsschemas wird der Kommunikationsbedarf sowohl im Einzel- als auch im Multicluster-Betrieb minimiert.

Die parallele Dendritensimulation gliedert sich in drei Komponenten: Simulationsanwendung, Parallelisierung und Lastausgleich. Kapitel 2 liefert einen Überblick über die zum weiteren Verständnis erforderlichen Grundlagen. Kapitel 3 erklärt, wie die Komponenten zu einer Anwendung zusammengeführt wurden. In Kapitel 4 wird die parallelisierungsspezifische Problemlösung beschrieben, während Kapitel 5 die Lösung der mit dem Lastausgleich verbundenen Probleme erklärt. Kapitel 6 präsentiert Laufzeitmessungen, die unter Einsatz der vorgestellten Algorithmen entstanden sind. Eine abschließende Betrachtung der erzielten Ergebnisse finden sich in Kapitel 7.

2 Grundlagen

Um das Verständnis der nachfolgenden Kapitel zu erleichtern, werden in den folgenden Abschnitten die Grundlagen der Dendritensimulation sowie der Parallelisierung und des Lastausgleichs beschrieben.

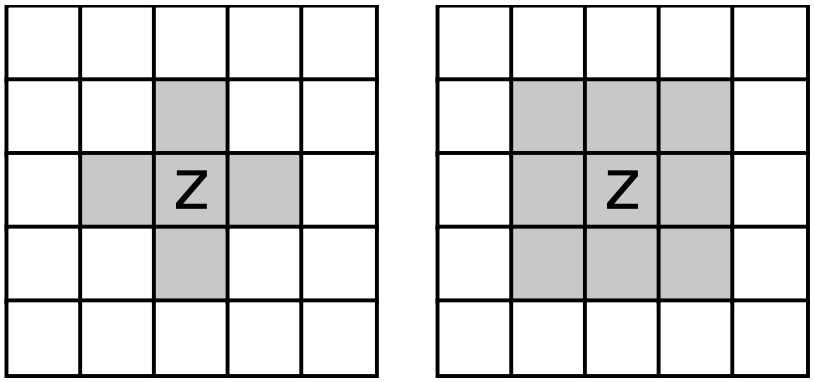
2.1 Simulationsanwendung

Die Simulation dendritisch erstarrender Metallschmelzen erfolgt mit einem *kontinuierlichen Automaten.* Da die Eigenschaften dieses Rechenmodells die Struktur des parallelen Algorithmus bestimmen, folgt zunächst eine allgemeine Funktionsbeschreibung. Anschließend wird ein Überblick über das zu Grunde liegende physikalische Modell gegeben.

2.1.1 Kontinuierliche Automaten

Kontinuierliche Automaten stellen eine Verallgemeinerung zellulärer Automaten dar. Als zellulären Automaten (CA) bezeichnet man ein universales Rechenmodell, das 1940 von Stanislaw Ulam und John von Neumann entwickelt wurde und seitdem verschiedene Anwendungen fand, insbesondere bei der Simulation physikalischer Phänomene [1, 2]. Grundlage eines CA ist eine reguläres Gitter von Zellen. Jede dieser Zellen befindet sich in einem Zustand $q \in Q, |Q| \in \mathbb{N}$. Der CA führt einen Rechenschritt aus, indem alle Zellen ihren Zustand synchron mittels einer Übergangsfunktion $f : Q^k \to Q$ neu bestimmen. Der neue Zustand einer Zelle hängt dabei ausschließlich vom Zustand der k Zellen in einer zuvor definierten Nachbarschaft ab. Abbildung 2.1 zeigt die *Neumann-* sowie die *Moore-Nachbarschaft* als Beispiele häufig verwendeter Nachbarschaften. Da in der Praxis nur endliche Gitter verarbeitet werden können, muss die Behandlung von Zellen am Gitterrand gesondert erfolgen. Eine gebräuchliche Methode ist es, die Randzellen konstant zu halten. Möglich ist auch eine zyklische Fortsetung der Nachbarschaft, beispielsweise mit einem torusförmigen Gitter.

Bei kontinuierlichen Automaten tritt an Stelle der endlichen Zustandsmenge eine unendliche Menge. Bei dem dieser Arbeit zu Grunde liegenden Automaten ist $Q = \mathbb{R}^n$.



(a) Neumann-Nachbarschaft (b) Moore-Nachbarschaft

Abbildung 2.1: Ausschnitte eines zweidimensionalen kartesischen Gitters. Die Nachbarschaft der Zelle Z ist jeweils dunkel hervorgehoben.

2.1.2 Physikalisches Modell

Die Qualität gussmetallener Werkstoffe schwankt in Abhängigkeit von externen Faktoren während des Erstarrungsprozesses. Um die Werkstoffqualität zu erhöhen, forscht das Jenaer Institut für Materialwissenschaft und Werkstofftechnik an physikalischen Modellen, die diesen Prozess beschreiben. Die computergestützte Simulation ist dabei ein wichtiges Mittel zur Modellverbesserung.

Im Rahmen dieser Arbeit ist eine Modelleigenschaft von besonderem Interesse: Der Rechenbedarf der Übergangsfunktion f schwankt in Abhängigkeit von Gitterkoordinate und Simulationszeitpunkt. Wie es dazu kommt, sei kurz erläutert. Das Modell teilt alle Zellen des Simulationsgitters in 3 Kategorien ein:

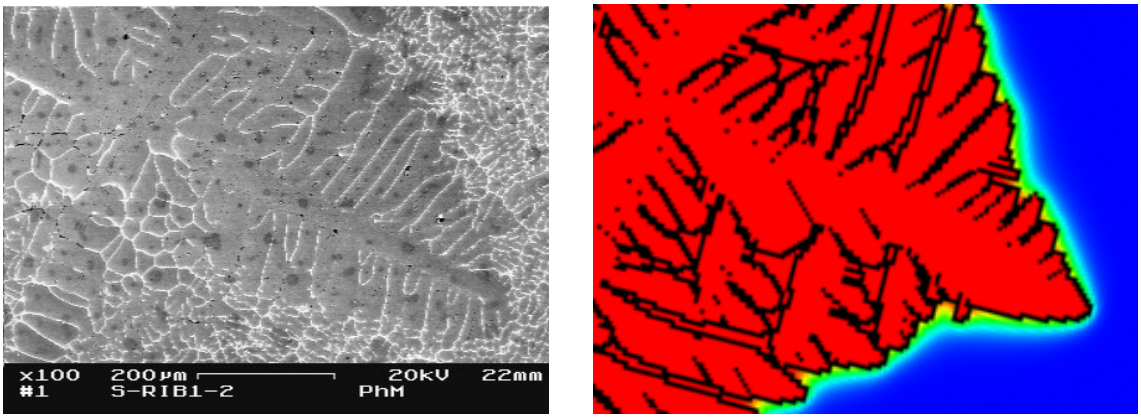
1. flüssige Zellen,

2. feste Zellen,

3. Zellen, in denen sich der Übergang von fester und flüssiger Phase, die so genannten Phasengrenze, befindet.

Die Berechnung der Positionsveränderung eben jener Phasengrenze ist der komplexeste Teil der Simulation. Dies hat zur Folge, dass die Übergangsfunktion f bei Zellen der letztgenannten Kategorie einen erheblich höheren Rechenbedarf verursacht. Da sich die Phasengrenze im Laufe der Zeit durch das Simulationsgebiet bewegt, entstehen örtlich und zeitliche begrenzte Berechnungsschwerpunkte *(Hotspots)*.

Das von der Phasengrenze umschlossene Gebiet erstarrter Schmelze nimmt baumartige Verzweigungsstrukturen an und wird als *Dendrit* bezeichnet. Abbildung 2.2 zeigt die Mikroskopaufnahme eines Al-Cu-Dendriten im Vergleich zum entsprechenden Simulationsergebnis.



(a) Mikroskopaufnahme (b) Simulationsergebnis

Abbildung 2.2: Mikroskopaufnahme eines Dendriten im Vergleich zum Simulationsergebnis. Der Verlauf der Phasengrenze ist im Simulationsbild schwarz markiert. Graustufen zeigen den Temperaturverlauf der teilweise erstarrten Metallschmelze an.

2.2 Parallele Zelluläre Automaten

Für die Parallelisierung von kontinuierlichen Automaten eignet sich ein datenparalleler Algorithmus, da die Übergangsfunktion für jede Zelle unabhängig von allen anderen Zellen ausgeführt werden kann. Die erweiterte Menge von Zellzuständen gegenüber normalen zellulären Automaten ist dabei unerheblich, so dass sich die Vorgehensweise bei beiden Automatentypen letztlich nicht unterscheidet. Parallele zelluläre Automaten (PCA) sind gut untersucht und ihre Umsetzung ähnelt üblicherweise dem folgenden Schema [3, 4, 5]:

1. Aufteilung des Simulationsgitters unter den Rechenknoten,
2. wiederholte Durchführung eines Simulationsschritts:
   a) Berechnung der Übergangsfunktion,
   b) Austausch von Randwerten,
3. Zusammenführung der Ergebnisse.

Die Aufteilung eines Simulationsschrittes in Berechnungs- und Kommunikationsphase entspricht den *Supersteps* des *Bulk Synchronous Parallel Model* [6]. Im Zusammenhang mit den im Abschnitt 2.2.3 diskutierten Optimierungen kann diese scharfe Trennung jedoch wieder abgeschwächt werden. Zunächst erfolgt jedoch eine genauere Betrachtung der Aufteilung und des Randwerteaustausches.

2.2.1 Aufteilung

Um die korrekte Ausführung des Automaten zu gewährleisten, muss jede Zelle und damit jede Gitterkoordinate genau einem Knoten zugewiesen werden. Sei C die Menge aller Koordinaten des Automatengitters und $P = (p_0, \ldots, p_n)$ die Liste aller Rechenknoten. Eine Aufteilung des Gitters, welche die obige Forderung erfüllt, wird mit D bezeichnet und ist folgendermaßen definiert:

$$D = (d_0, \ldots, d_n), \quad \bigcup_i d_i = C, \quad \forall_{i \neq j} \, (d_i \cap d_j = \emptyset) \tag{2.1}$$

Dabei ist d_i die Menge aller dem Knoten p_i zugewiesenen Koordinaten. Wegen der direkten Zuordnung zwischen Zellen und ihren Koordinaten wird D im Sprachgebrauch sowohl als Aufteilung von Zellen als auch von Koordinaten bezeichnet. Die praktische Berechnung von Aufteilungen wird in Abschnitt 4 behandelt.

2.2.2 Randwerteaustausch

Um den Zustand einer Zelle z im Simulationsschritt $t+1$ zu berechnen, benötigt die Übergangsfunktion f den Zustand aller Nachbarzellen von z. Die genaue Form dieser Nachbarschaft kann je nach Anwendung variieren. Sei $\text{neighbors}(c) : C \to 2^C$ die Funktion, welche der Koordinate c alle benötigten Nachbarkoordinaten zuordnet. Damit ein Rechenknoten p_i mit der Koordinatenmenge d_i die Übergangsfunktion für alle seine Zellen berechnen kann, müssen p_i also die Zustände aller Zellen in einer erweiterten Koordinatenmenge d_i^+ bekannt sein.

$$d_i^+ = \bigcup_{c \in d_i} \text{neighbors}(c) \tag{2.2}$$

Die Zellen der Koordinaten $d_i^+ \setminus d_i$, des sogennanten *äußeren Randes* von d_i, müssen ihm dafür von anderen Knoten zugesandt werden. Entsprechend muss der Knoten p_i alle Zellen des *inneren Randes* $(\bigcup_{j \neq i} d_j^+) \cap d_i$ an andere Knoten senden. Bei vielen Gittertypen existieren Koordinaten des äußeren Randes, die selbst nicht Teil des Simulationsgebiets $C = \bigcup_i d_i$ sind. Diese bilden eine Ausnahme und werden beim Randwerteaustausch nicht beachtet. Die Menge aller Koordinaten von Zellen, die beim Randwerteaustausch versendet werden müssen, bezeichnet man als *Oberfläche* S_i eines Knotens, bzw. als *Oberfläche* S der Aufteilung, wenn alle Knoten gemeinsam betrachtet werden. Abbildung 2.3 zeigt den inneren und äußeren Rand am Beispiel eines 4×6-Gitters bei Anwendung der Neumann-Nachbarschaft. Die Zellen des Gebiets d_i sind in hellem Grau hervorgehoben. Hell- und dunkelgraue Zellen bilden zusammen das Gebiet d_i^+.

2.2.3 Optimierungen

Die folgenden Leistungsoptimierungen für Datenparallele Algorithmen auf regulären Gittern sind bei Mattson et. al. [7] beschrieben und kommen für die Optimierung von PCA in Betracht:

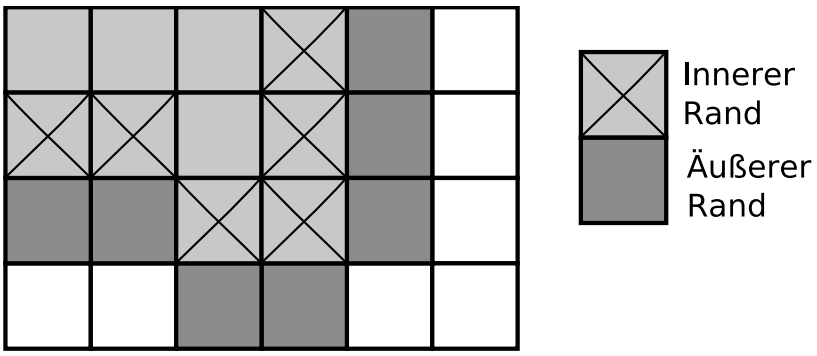


Abbildung 2.3: Ränder des hellgrau gekennzeichneten Gebiets d_i.

Oberflächenoptimierung Der Kommunikationsbedarf beim Randwerteaustausch hängt direkt von der Oberfläche der Zellmenge eines jeden Rechenknotens ab. Die Wahl einer geeigneten Aufteilung kann diese Oberfläche reduzieren.

Überlappung von Berechnung und Kommunikation Die Zellen eines Knotens lassen sich in zwei Klassen einordnen. Solche, die beim Randwerteaustausch versandt werden müssen, und alle übrigen. Werden Randzellen zuerst berechnet, kann der Randwerteaustausch parallel zur Berechnung aller inneren Zellen erfolgen. Der Gewinn beim Einsatz dieser Optimierung hängt stark vom Verhältnis der Gittergröße zur Anzahl der Rechenknoten ab [8].

Nachrichtenbündelung Der Durchsatz von Netzwerkverbindungen steigt bis zu einem Sättigungspunkt mit der Größe der versandten Nachrichten. Deswegen ist es sinnvoll, anstelle vieler kleiner Nachrichten wenige große Nachrichten zu versenden. Um dies zu erreichen, sollten alle bei einem Randwerteaustausch involvierten Zellen gebündelt versandt werden. Wird mit dieser Technik der Sättigungspunkt nicht erreicht, können die versandten Nachbarschaften k-fach vergrößert werden, so dass ein Randwerteaustausch nur noch alle k Rechenschritte, dafür aber mit größeren Nachrichten erfolgt. Die letztgenannte Technik wird als *Ghosting* bezeichnet.

2.2.4 Multicluster

Die Parallelisierung serieller Algorithmen entspringt dem Wunsch, die Ausführungsgeschwindigkeit mit steigender Anzahl eingesetzter Rechenknoten zu erhöhen. Reicht die Anzahl der in einem Rechencluster vorhandenen Knoten nicht aus, bietet der Verbund mehrerer Cluster zu einem Multi-Domain-Cluster, kurz Multicluster, einen möglichen Ausweg. Die jüngsten Fortschritte auf dem Gebiet der Clusterkopplung, beispielsweise mit IPv6 und Open MPI [9], erleichtern diese Vorgehensweise. Die bei der Kopplung von Clustern eingesetzten Netzwerke sind üblicherweise weniger leistungsstark als die clusterinternen Kommunikations-

verbindungen. Um den Einsatz von Multiclustern auch für kommunikationsintensive Anwendungen zu ermöglichen, muss diese Heterogenität der Infrastruktur beim Algorithmenentwurf berücksichtigt werden. Wo immer möglich, sollte Kommunikation über Clustergrenzen hinweg vermieden werden.

Ein weiterer Aspekt beim Übergang vom Einzel- zum Multicluster ist das Anwachsen der Knotenzahl. Bei der Parallelisierung von PCA wird jedoch in Abhängigkeit von der Problemgröße ein Sättigungspunkt erreicht, ab dem zusätzliche Knoten keine Beschleunigung erzielen. Vom Hinzufügen weiterer Knoten kann die parallele Berechnung einer konkreten Probleminstanz also nur profitieren, wenn der leistungsstärkste Einzelcluster diesen Sättigungspunkt noch nicht erreicht.

2.3 Lastausgleich

Eine Darstellung der Lastausgleichsproblematik lässt sich allgemein in die folgenden Unterpunkte gliedern:

1. Warum ist ausgeglichene Last wünschenswert?
2. Was sind die Ursachen für unausgeglichene Last?
3. Wie gelangt man vom unausgeglichenen in den ausgeglichenen Zustand?
4. Wann ist Lastausgleich zweckmäßig?

Die konkrete Beantwortung der letzten beiden Fragen liefert einen Lastausgleichsalgorithmus.

Die gleichmäßige Auslastung aller Rechenknoten ist eine notwendige Bedingung für die Effizienz einer parallelen Berechnung. PCA sind Beispiele für *synchrone* Algorithmen: Knoten, die für die Berechnung benachbarter Zellen zuständig sind, müssen nach jedem Rechenschritt Randwerte austauschen. Eine gleichmäßige Auslastung kann demzufolge nur gewahrt bleiben, wenn alle Knoten für jeden Rechenschritt die gleiche Zeit benötigen. Andernfalls dominiert der jeweils langsamste Knoten die Ausführungszeit, während andere Knoten auf Randwerte warten.

Gründe für Variationen in den Berechnungszeiten einzelner Knoten können sowohl statischer als auch dynamischer Natur sein. Werden PCA in einer unbekannten, heterogenen Umgebung ausgeführt, ist es üblich, das Zellgitter gleichmäßig unter den Knoten aufzuteilen. Aus den unterschiedlichen Leistungsdaten der beteiligten Rechenknoten ergibt sich dann ein statisches Ungleichgewicht.

Dynamisch schwankende Last tritt auf, wenn die Knoten nicht exklusiv für die Berechnung des PCA zur Verfügung stehen, sondern zeitgleich von anderen Prozessen genutzt werden. Auch die Übergangsfunktion f kann Quelle dynamischen Ungleichgewichtes sein, wenn statt eines klassischen zellulären Automaten eine verallgemeinerte Variante verwendet wird. Bei einer kleinen Zustandsmenge Q ist

es oft möglich, f mittels einer Wertetabelle zu implementieren. Diese Art der Funktionsberechnung weist eine konstante Geschwindigkeit auf und verursacht daher auch kein Ungleichgewicht. Handelt es sich bei Q jedoch um die quasi-unendlichen Menge von Gleitkommazahlen, kommt eine Wertetabelle nicht mehr in Frage. Wenn der Aufwand zur Berechnung von f in Abhängigkeit vom Zellzustand variiert, kann es zur Bildung von Hotspots, Gebieten mit überdurchschnittlicher Rechenlast, kommen. Dieses Phänomen ist insbesondere bei der in Abschnitt 2.1.2 vorgestellten Dendriten-Simulation zu beobachten.

Nachdem mit der Beantwortung der Fragen 1 und 2 das Problem unausgeglichener Lasten skizziert wurde, folgt nun die Behandlung von Punkt 3: der Probemlösung. Das Gleichgewicht zwischen den Rechenknoten lässt sich durch die Migration von Arbeitslast wieder herstellen. Bei PCA werden dafür Zellen von überlasteten auf unausgelastete Knoten verschoben. Diese Einteilung der Knoten beruht auf Lastmessungen, die periodisch vorgenommen werden. Findet die Lastmigration wiederholt während der Ausführung der Anwendung statt, spricht man von *dynamischem* Lastausgleich (DLB).

Das Verschieben von Zellen entspricht dem Übergang von der Aufteilung D_{alt} zu einer neuen Aufteilung D_{neu}. Es wird also ein Algorithmus benötigt, der aus dem gemessenen Lastzustand des Systems eine besser balancierte Aufteilung D_{neu} errechnet. Die Menge aller Aufteilungen, die von solch einem Algorithmus erzeugt werden können, bezeichnet man als *Aufteilungsschema*. Üblicherweise beinhaltet diese Menge nur einen Bruchteil aller theoretisch möglichen Aufteilungen. Es lassen sich daher spezifische Forderungen an einen Algorithmus und das erzeugte Schema stellen: Einerseits soll die Oberfläche S aller Aufteilungen möglichst gering sein, andererseits ist eine hohe *Überlappung* von D_{alt} und D_{neu} gewünscht. Als Überlappung bezeichnet man die Menge der Koordinaten, die nicht migriert werden müssen, d. h. $\bigcup_i (d_{\text{alt}_i} \cap d_{\text{neu}_i})$. Zu diesen schwer vereinbarenden Forderungen kommt es, weil einerseits jede Lastmigration so unaufwendig wie möglich sein soll, andererseits aber auch der Kommunikationsaufwand beim Randwerteaustausch nicht überhand nehmen darf. Eine detaillierte Auseinandersetzung mit Aufteilungsalgorithmen und den dazugehörigen Aufteilungsschemata erfolgt in Abschnitt 4.2.

Ein wichtiger Zusammenhang besteht zwischen dem Aufteilungsalgorithmus und der Architektur des DLB-Algorithmus. Ist sichergestellt, dass jede Lastmigration nur eine begrenzte Zahl von Knoten betrifft, so kann der Lastausgleich *lokal* erfolgen. Dies hat Implikationen für die Erhebung von Messdaten und den Kommunikations- und Rechenaufwand des DLB-Algorithmus. Wenn die Anzahl betroffener Knoten nicht beschränkt werden kann, sollte hingegen ein *zentraler* Algorithmus gewählt werden. Eine Beschreibung lokaler und zentraler DLB-Algorithmen findet sich bei Plastino et. al. [10].

Ebenso wichtig wie die Wahl einer neuen Gitteraufteilung ist die Wahl eines geeigneten Zeitpunktes für die Durchführung des Lastausgleichs. Da während einer Migration die Berechnung des PCA ausgesetzt werden muss, ist die erhoffte Zeitersparnis gegen diesen Zeitverlust abzuwägen. Damit sind wir bei Punkt 4 ange-

langt: der Frage nach der Zweckmäßigkeit des Lastausgleichs. Zur Lösung dieses Problems sind Algorithmen unterschiedlichsten Komplexitätsgrades beschrieben worden. Eine Gemeinsamkeit findet sich jedoch in ihrer Abhängigkeit von Parametern, deren korrekte Einstellung über die Effizienz des Verfahrens entscheidet. Ein kurzer Überblick möge dies verdeutlichen: Ausgehend von der Taxonomie von Plastino et. al. können Lastausgleichsalgorithmen anhand ihres Aktivierungsmechanismus als *periodisch* oder *ereignisgetrieben* beschrieben werden. Periodische Algorithmen führen eine Lastmigration alle k Rechenschritte durch und sind dadurch einfach zu implementieren. Der Nutzen dieser Vorgehensweise hängt stark davon ab, ob ein geeigneter Wert für den Parameter k gefunden werden kann [10]. Ein zu geringer Wert für k vergeudet Zeit durch unnötige Migrationen, während ein zu hoher Wert unausgeglichene Last nicht schnell genug beseitigt. Ändert sich die Geschwindigkeit der Lastschwankungen fortwährend, ist Ineffizienz unvermeidbar. Die Abhängigkeit vom Parameter k reduziert sich bei ereignisgetriebenen Algorithmen durch die Einführung eines zusätzlichen Schwellwertparameters t. Alle k Schritte wird die Unausgewogenheit der Arbeitslast im System quantifiziert und mit t verglichen. Nur eine Überschreitung von t führt zur Lastmigration. Bei geeigneter Wahl von t wird unnötiger Lastausgleich vermieden. Die Festlegung von t kann auf Basis von Testläufen erfolgen [11]. Eine mögliche Alternative zur manuellen Wahl von t bietet das Verfahren von Ka-Yeung Kwok et. al. [12]. Dabei werden sowohl die erwarteten Zeitersparnisse als auch die Kosten einer Zellmigration dynamisch aus Laufzeitdaten geschätzt. Die geschätzten Kosten liefern den Schwellwert, mit denen die erwarteten Zeitersparnisse verglichen werden können. Wegen des Verzichts auf einen experimentell zu wählenden Schwellwertparameter bietet der letztgenannte Algorithmus die besten Voraussetzungen, um die Anforderungen eines sich wandelnden Simulationsmodells zu erfüllen.

3 Architektur

Die im Rahmen der vorliegenden Arbeit entwickelten Algorithmen sind eingebettet in die parallele Dendritensimulation *MuCluDent*. Um die Einordnung der im Folgenden diskutierten Komponenten zu erläutern, soll ein Überblick über die Architektur dieser Software gegeben werden. *MuCluDent (Multi-Cluster-Dendrites)* wird von der Arbeitsgruppe Cluster- & Meta-Computing in Kooperation mit dem Institut für Materialwissenschaft und Werkstofftechnik entwickelt. Es ist darauf ausgelegt, die Berechnung des ständig weiterentwickelten physikalischen Modells durch Einsatz von Clustern und Multiclustern zu beschleunigen.

MuCluDent ist für die Verwendung im Batch-Betrieb vorgesehen. Der Nutzer legt zunächst eine Konfigurationsdatei an, in der Dimensionen und Startbelegung des Simulationsgitters, die gewünschte Anzahl von Simulationsschritten sowie diverse physikalisches Modellparameter festgelegt sind (siehe Anhang A). Diese Konfigurationsdatei wird beim Start von *MuCluDent* als Parameter übergeben. Ein weiterer Parameter ist die Ausgabeperiode o. Alle o Simulationsschritte wird der Zustand des gesamten Simulationsgitters aggregiert und in eine Datei geschrieben. Nach Beendigung der Simulation kann diese Datei einem Visualisierungsprogramm übergeben werden, um die Veränderung des Gitters über den Simulationszeitraum hinweg zu betrachten.

Die Architektur von *MuCluDent* folgt objektorientierten Prinzipien und spiegelt die in Abschnitt 2 beschriebenen Komponenten wider.

Cell Diese Klasse kapselt alle Elemente des physikalischen Modells. Eine Instanz dieser Klasse beschreibt den Zustand einer Zelle des kontinuierlichen Automaten und stellt die Übergangsfunktion zur Verfügung.

Grid Die Datenstruktur *Grid* verwaltet die Zellen des Automaten. Zum gegenwärtigen Zeitpunkt ist ein zweidimensionales kartesisches Gitter implementiert, aber eine Dimensionserweiterung ist möglich.

Partition Diese Klasse kapselt die Repräsentation aller Aufteilungen eines Aufteilungsschemas sowie einen Algorithmus zu deren Erzeugung. Die Implementierung dieser Klasse wird in Abschnitt 4 beschrieben.

LoadModel & CommunicationModel Diese Komponenten steuern den dynamischen Lastausgleich. Eine ausführliche Behandlung findet sich in Abschnitt 5.

Simulator Die Programmkomponente Simulator verwaltet das Simulationsgebiet eines Rechenknotens. Es synchronisiert die Überführung der Zellen in den

jeweiligen Folgezustand und führt den Randwerteaustausch zwischen den Knoten durch. Die notwendigen Kommunikationsvorgänge werden dynamisch aus der vorliegenden *Partition* generiert. Der Simulator liefert außerdem Laufzeitdaten an die Komponenten des Lastausgleichsalgorithmus. Wenn auf Basis dieser Daten eine Balancierung sinnvoll erscheint, koordiniert der Simulator die Migration der Zellen. Bei den anfallenden Kommunikationsvorgängen kommt das *Message Passing Interface (MPI)* [13] zum Einsatz.

Writer Ziel einer Simulation ist es, den Zustand des Simulationsgitters zu verschiedenen Zeitpunkten zu ermitteln. Die Komponente *Writer* schreibt die gewünschte Repräsentation des Gitters periodisch auf den Massenspeicher. Neben der verlustfreien Ausgabe des Gitterzustandes als MPI-Serialisierung (zur Betrachtung mit einem separaten Programm) wird die direkte Ausgabe verschiedener graphischer Formate unterstützt.

Die Klassen *Cell*, *Grid* und *Writer* betreffen den anwendungsspezifischen Teil des Programms. *Simulator* und *Partition* bilden die Hauptbestandteile der Parallelisierung, *LoadModel* und *CommunicationModel* schließlich sind für den Lastausgleich zuständig. Abbildung 3.1 zeigt das Zusammenspiel der genannten Komponenten. Die Klassen *Simulator, Partition, LoadModel* und *CommunicationModel* bilden die Implementierung der in dieser Arbeit entwickelten Algorithmen. Eine genaue Aufschlüsselung der Programmquellen von *MuCluDent* findet sich in Anhang C.

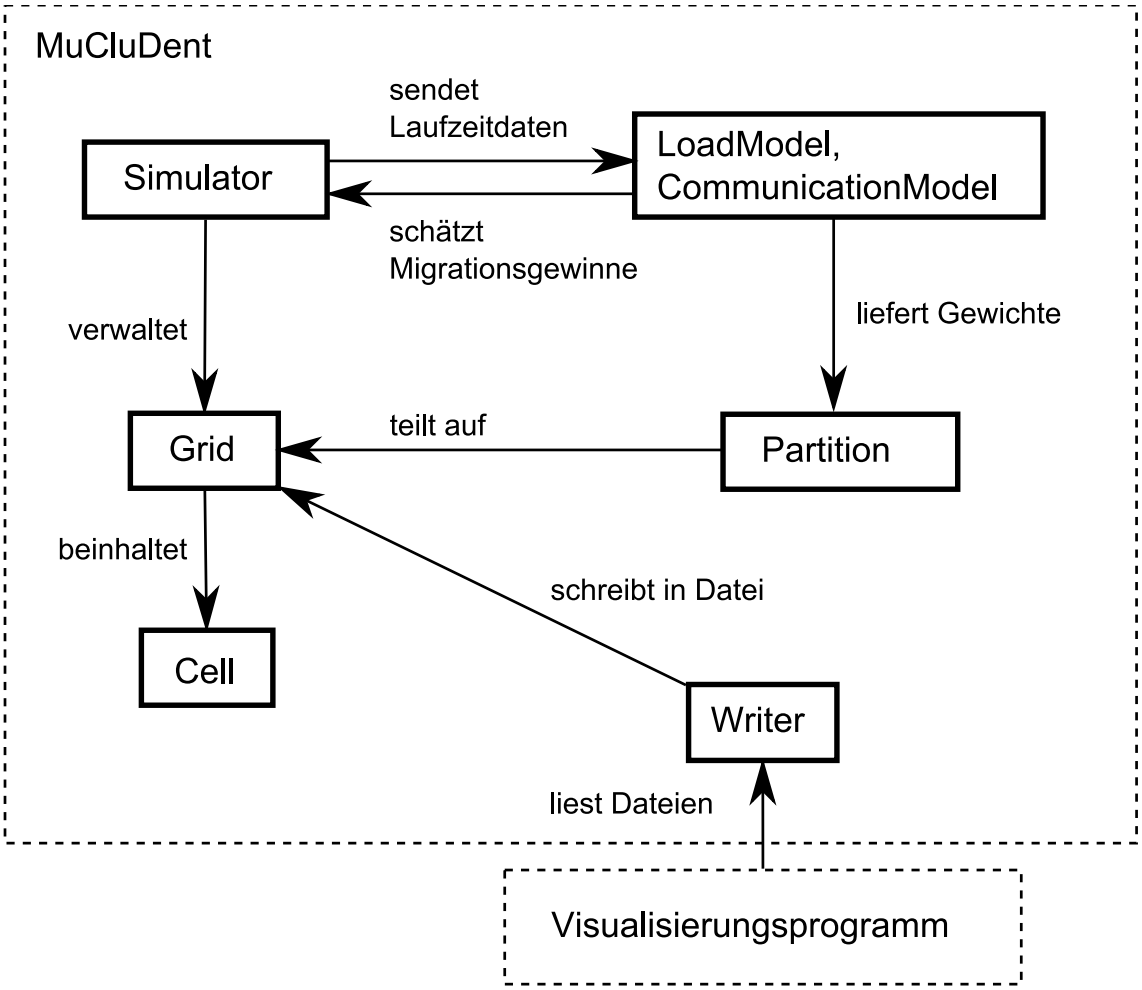


Abbildung 3.1: Zusammenhang der wichtigsten Komponenten von *MuCluDent* und ihr Zusammenspiel mit dem Visualisierungsprogramm.

4 Parallelisierung

Die Hauptaufgaben der Parallelisierung bestehen in der Aufteilung des Gitters sowie der Koordination des Randwerteaustausches. Die wichtigste Entscheidung, die dafür getroffen werden muss, ist die Wahl des Aufteilungsalgorithmus, da dieser das Aufteilungsschema bestimmt. Die Implementierung des Randwerteaustausches kann dann angepasst an dieses Schema erfolgen.

Abschnitt 4.1 befasst sich mit den genauen Anforderungen, die an das Aufteilungsschema gestellt werden. Danach folgt eine Darstellung gleichmäßiger Aufteilungen mit den Algorithmen *Striping, Checkerboard, Space Filling Curves (SFC)* und *Recursive Coordinate Bisection (RCB)*. Anschließend wird die Erweiterbarkeit der Algorithmen zur Erzeugung gewichteter Aufteilungen untersucht. Die Wahl des (auf *RCB* basierenden) Algorithmus *Weighted Recursive Coordinate Bisection (WRCB)* ergibt sich aus dem Vergleich aller erzeugten Aufteilungsschemata anhand der in Abschnitt 4.1 genannten Kriterien. Die Beschreibung der Implementierung von *WRCB* schließt sich an. Dabei wird auch auf den Randwerteaustausch und einige Optimierungsmöglichkeiten eingegangen.

4.1 Anforderungen

Wie in Abschnitt 2.3 dargestellt, muss der Aufteilungsalgorithmus eine balancierte Aufteilung für beliebige Lastzustände erzeugen können. Sei $|C|$ die Anzahl aller Zellen eines Gitters und $|d_i|$ die Anzahl der Zellen des Knotens p_i bei Aufteilung D. Für jeden Gewichtsvektor $W = (w_0, \ldots, w_n), w_i \in [0,1], \sum w_i = 1$ muss eine Aufteilung D erzeugbar sein, so dass gilt:

$$\forall_i (w_i \approx \frac{|d_i|}{|C|}) \tag{4.1}$$

Weiterhin muss die Oberfläche S eines so erzeugten D, und somit auch der Kommunikationsbedarf beim Randwerteaustausch, möglichst niedrig sein. Dieses Kriterium ist wichtig für den Speedup der parallelen Anwendung bei konstanter Problemgröße.

Die Überlappung zweier Aufteilungen $D_{\text{alt}}, D_{\text{neu}}$ bei gegebenen Gewichtsvektoren $W_{\text{alt}}, W_{\text{neu}}$ bestimmt den Kommunikationsbedarf bei der Lastmigration. Da die Migration jedoch im Vergleich zum Randwerteaustausch viel seltener stattfindet, ist die Forderung nach möglichst hoher Überlappung von untergeordneter Wichtigkeit.

Um ein Aufteilungsschema an die Struktur einer Multiclusterumgebung anzupassen, kommen die eben genannten Anforderungen in leicht verändertem Kontext erneut zum Tragen. Sei $\{cluster_0, \ldots, cluster_{m-1}\}$ mit $cluster_i \subseteq P$ eine Zerlegung der Knotenmenge P in m Cluster. Dann lässt sich analog zum einzelnen Knoten für jeden Cluster ein äußerer Rand und eine Oberfläche definieren. Die vereinigte Oberfläche aller Cluster, die *Inter-Cluster-Oberfläche,* muss minimiert werden.

Die Forderung nach möglichst hoher Überlappung der Clustergebiete beim Übergang von D_{alt} nach D_{neu} überträgt sich analog.

4.2 Aufteilungsalgorithmen

Bei der nun folgenden Betrachtung verschiedener Aufteilungsalgorithmen wird die jeweils zu Grunde liegende Idee zuerst am Beispiel einer gleichmäßigen Aufteilung erläutert. Anschließend wird dann gezeigt, wie sich diese Idee zu einer gewichteten Aufteilung verallgemeinern lässt.

Zur vergleichenden Oberflächenberechnung wird ein zweidimensionales kartesisches Gitter herangezogen. Um den gebräuchlichsten Anwendungsfall widerzuspiegeln, wird dieses Gitter als quadratisch vorausgesetzt. Die Berechnung vereinfacht sich weiter, indem an Stelle eines diskreten Gitters das Einheitsquadrat aufgeteilt wird. Der reellwertige Umfang der Knotengebiete verhält sich in guter Näherung proportional zum tatsächlichen Kommunikationsbedarf. Der Umfang des Einheitsquadrates wird vom Gesamtumfang abgezogen, da bei *MuCluDent* nicht über die Gitterränder kommuniziert werden muss.

4.2.1 Striping

Die Aufteilung des Gitters entlang einer Koordinatenachse wird als Striping bezeichnet. Je nach Wahl der Achse entstehen rechteckige Streifen, welche die gesamte Höhe beziehungsweise Breite des Gitters einnehmen.

Bei wachsender Knotenzahl werden die resultierenden Rechtecke zunehmend schmaler und das Verhältnis von Umfang zu Flächeninhalt, der *Formfaktor,* verschlechtert sich. Die Kommunikationsoberfläche in Abhängigkeit von der Knotenzahl n ergibt sich als:

$$\mathrm{S}_{\text{striping}}(n) = 2n - 2 \tag{4.2}$$

Die Verallgemeinerung des Striping-Algorithmus zur Erzeugung gewichteter Aufteilungen erfolgt durch Variation der Streifenbreite. Bei der Aufteilung des Einheitsquadrates gemäß des Gewichtsvektors W erhält der i-te Streifen die Breite w_i. Die so gewichtete Aufteilung hat dieselbe Oberfläche wie die ausgewogene Striping-Aufteilung. Abbildung 4.1 zeigt die gewichtete Aufteilung des Einheitsquadrates unter 4 Knoten.

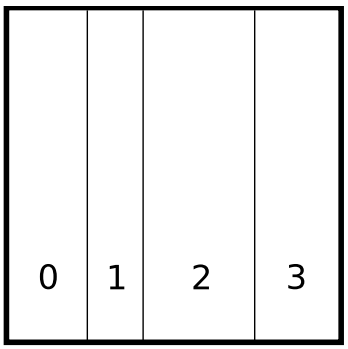


Abbildung 4.1: Gewichtete Striping-Aufteilung des Einheitsquadrates unter 4 Knoten.

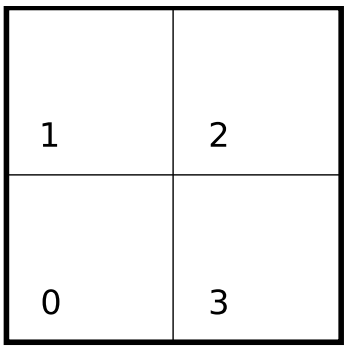


Abbildung 4.2: Checkerboard-Aufteilung des Einheitsquadrates unter 4 Knoten.

4.2.2 Checkerboard

Im Gegensatz zu Striping wird das Gitter bei der Checkerboard-Aufteilung entlang aller Koordinatenachsen aufgeteilt. Lässt sich die Anzahl der Knoten n so faktorisieren, dass die Faktoren $f_i, n = \prod f_i$ dem Verhältnis der Gitterdimensionen entsprechen, erzeugt diese Aufteilung quadratische Gebiete. Abbildung 4.2 zeigt diese Aufteilung im Fall $n = 4$.

Das Einheitsquadrat lässt sich also auf diese Weise aufteilen, wenn n eine Quadratzahl ist. Die Oberfläche berechnet sich in diesem Fall als:

$$\mathrm{S}_{\mathrm{checkerboard}}(n) = 4 \cdot \sqrt{n} - 4 \tag{4.3}$$

Die quadratische Form bietet bei Rechtecken mit konstanten Flächeninhalten den optimalen Formfaktor. Da bei vielen Kombinationen von Gitterdimensionen und Knotenzahl nur eine Näherung an die quadratische Form erreicht wird, liefert die obige Formel im allgemeinen Fall lediglich eine untere Schranke. Ist die ideale Quadratform realisierbar, wird mit einer gleichmäßigen Checkerboard-Aufteilung die minimal mögliche Oberfläche realisiert [14].

Unglücklicherweise lässt sich diese vorteilhafte Aufteilung nicht auf den gewichteten Anwendungsfall verallgemeinern. Wird die Aufteilung entlang einer Koordinatenachse variiert, ändern sich die Größen mehrerer Gebiete auf einmal. Die unabhängige Größenanpassung aller Gebiete ist somit unmöglich.

4.2.3 Space Filling Curves (SFC)

Als Space Filling Curves (SFC) bezeichnet man stetige Funktionen die einem eindimensionalen Definitionsbereich surjektiv auf einen mehrdimesionalen Wertebereich abbilden [15]. Sie können zur örtlichen Gruppierung von Gitterkoordinaten und damit zur Berechnung von Aufteilungen mit geringer Oberfläche eingesetzt werden. Eine diskrete SFC ordnet jeder Koordinate einen Index zu. Dabei ist die räumliche Distanz zweier Koordinaten stark mit der arithmetischen Distanz ihrer Indices korreliert. Werden nun Sequenzen aufeinanderfolgender Indices einem Knoten zugewiesen, so hat das resultierende Gebiet meist einen guten Formfaktor.

Ein Zahlenbeispiel für die Oberfläche von SFC-Aufteilungen ist der Arbeit von Wierum [14] entnommen. Die obere Schranke der durchschnittlichen Oberfläche bei Verwendung der Lebesgue-Kurve ist dort als Vielfaches der Optimaloberfläche gegeben:

$$S_{SFC}(n) = \frac{10}{\sqrt{3}}\sqrt{n} - 4 \tag{4.4}$$

Eine Verallgemeinerung dieser Vorgehensweise zu Erzeugung gewichteter Aufteilungen ist realisierbar, indem man die Länge der Indexsequenzen proportional zu den vorliegenden Gewichten W setzt.

Wie in der Arbeit von Goldau [16] gezeigt wird, kann die Oberfläche einer gewichteten SFC-Aufteilung stark vom Mittelwert abweichen. Das Risiko einer degenerierten Oberfläche wiegt umso stärker, da es auch die Inter-Cluster-Oberfläche betrifft.

4.2.4 Recursive Coordinate Bisection (RCB)

Der von Berger et. al. beschriebene RCB-Algorithmus zerlegt ein Gitter nach dem *Teile-und-Herrsche*-Muster [17]. Soll ein Gitter unter $n = a + b$ Knoten aufgeteilt werden, so teilt man es zunächst entlang einer Koordinatenachse im Verhältnis a zu b. Es verbleibt die Aufteilung kleinerer Gitter unter a beziehungsweise b Knoten. Die rekursive Anwendung dieser Vorgehensweise endet bei der trivialen Aufteilung eines Gitters auf einen einzigen Knoten. Wählt man bei jeder Teilung die Koordinatenachse so, dass das Gitter entlang seiner längsten Dimension aufgeteilt wird und setzt man außerdem $a \approx \frac{n}{2}$, so ergeben sich Gebiete mit einem guten Formfaktor. Dies wird deutlich, wenn man das ungünstigste Seitenverhältnis eines bei einer Teilung erzeugten Rechtecks betrachtet. Dieses extreme Seitenverhältnis von 1:3 tritt bei der Teilung eines Quadrates unter drei Knoten auf. Werden hingegen Rechtecke mit einem schlechteren Formfaktor geteilt, so ergibt sich automatisch ein besseres Seitenverhältnis. Abbildung 4.3 zeigt den Rekursionsbaum bei der gleichmäßigen Aufteilung des Einheitsquadrates unter 4 Knoten.

Der genaue Zusammenhang zwischen Oberfläche S_{RCB} und Knotenzahl n wurde experimentell ermittelt. Die Ergebnisse dieser Untersuchung sind in Abbildung 4.4 wiedergegeben.

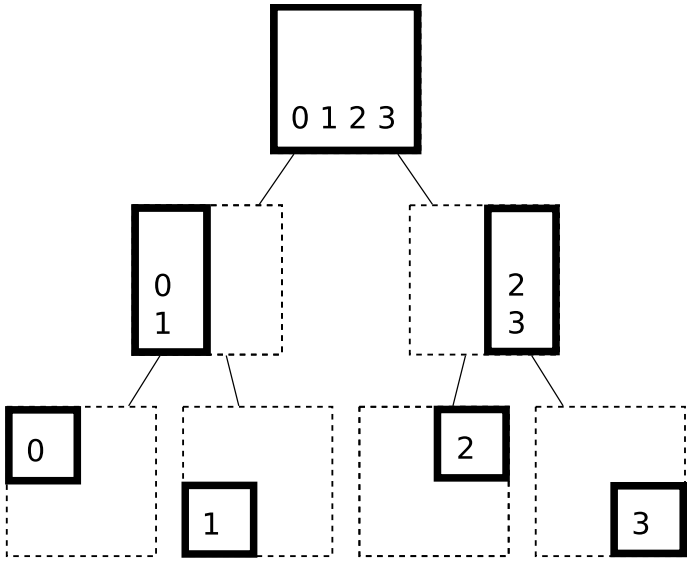


Abbildung 4.3: Rekursionsbaum bei gleichmäßigen Aufteilung des Einheitsqadrates mit dem RCB-Algorithmus. In das jeweils verbleibende Rechteck (durchgezogen umrahmt) sind die Indices der zugeordneten Knoten eingetragen.

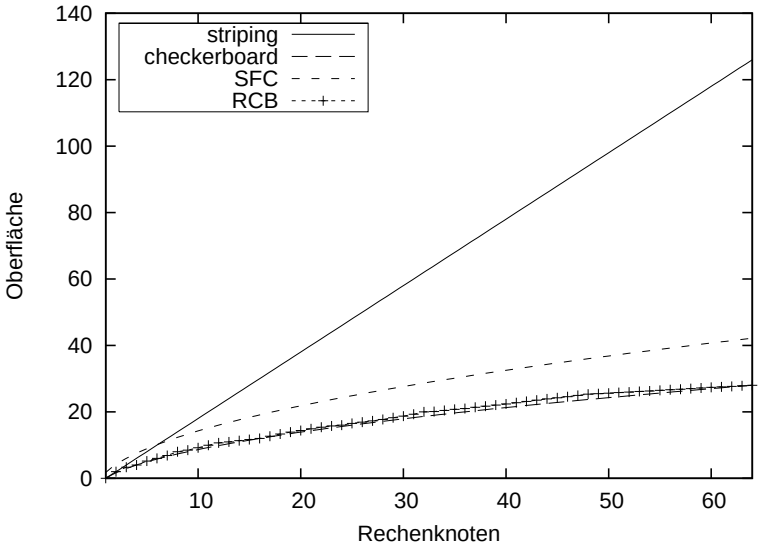
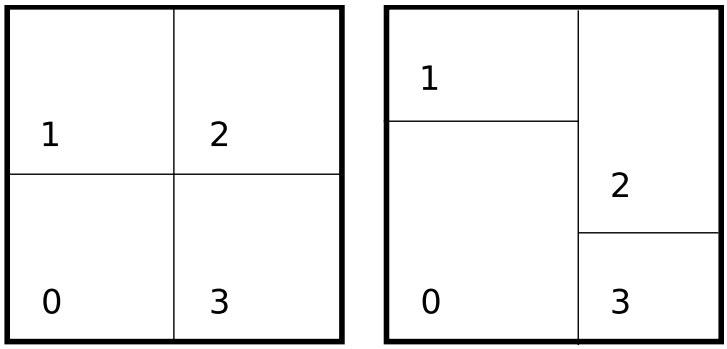


Abbildung 4.4: Vergleich der Kommunikationsoberfläche bei gebietsgleicher Aufteilung eines quadratischen Gitters mit unterschiedlichen Verfahren.



(a) gleichmäßige Aufteilung (b) gewichtete Aufteilung

Abbildung 4.5: Aufteilung des Einheitsquadrates mit RCB- und WRCB-Algorithmus.

Die Verallgemeinerung von RCB auf den gewichteten Anwendungsfall ist wie folgt realisierbar: Bei jeder Teilung eines Rechtecks zwischen den Knoten $p_0, \ldots, p_n$ in die Gruppen $A = \{p_0, \ldots, p_{a-1}\}, B = \{p_a, \ldots, p_n\}$ summiert man die Gewichte W_A, W_B beider Seiten

$$W_A = \sum_{i:\, p_i \in A} w_i \qquad (W_B \text{ entsprechend}) \tag{4.5}$$

und teilt im Verhältnis W_A zu W_B. Abbildung 4.5 zeigt die gewichtete Aufteilung des Einheitsquadrates im Vergleich zur gleichmäßigen Aufteilung.

Der Einfluss schwankender Gebietsgrößen auf die Oberfläche ist im Einzelfall schwer vorhersehbar. Die Aufteilung mit gewichtetem RCB (WRCB) kann Gebiete erzeugen, deren Form stärker von der idealen Quadratform abweicht als im ungewichteten Fall. Andererseits tauchen Gebiete auf, deren Größe über dem Durchschnitt liegt und die damit vom allgemein besseren Formfaktor größerer Rechtecke profitieren. Dieser Größenbonus ergibt sich aus den unterschiedlichen Wachstumsordnungen von Umfang und Flächeninhalt bei proportionaler Vergrößerung von Flächen.

Um den Gesamteffekt abzuschätzen, wurde die Kommunikationsoberfläche des Einheitsquadrates bei Aufteilung unter n Knoten und zufälligen Gewichten gemessen. Die Gewichte der einzelnen Knoten stammen gleichverteilt aus dem Intervall $[0,1]$. Abbildung 4.6 zeigt den Mittelwert der 2^{17}-fach wiederholten Messung. Wie man sieht, gleichen sich die oben aufgeführten Effekte gegenseitig aus, so dass sich die Oberfläche im Durchschnitt sowie im Maximalfall kaum von der einer ungewichteten RCB-Aufteilung unterscheidet (siehe Abbildung 4.6). Auch wenn dieses Experiment eine formale Worst-Case Analyse nicht ersetzen kann, so ist doch gezeigt, dass die Wahrscheinlichkeit einer signifikanten Oberflächendegeneration äußerst gering ausfällt.

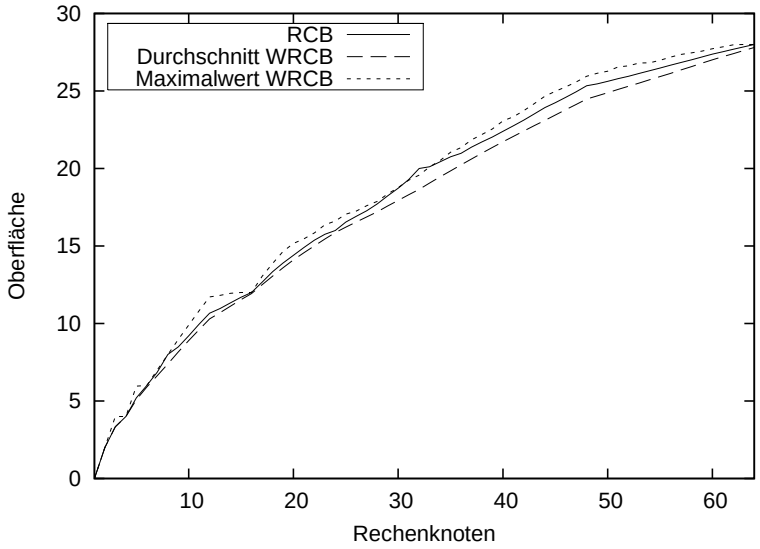


Abbildung 4.6: Vergleich der Oberfläche bei gleichmäßiger RCB-Aufteilung mit durchschnittlicher und maximaler Oberfläche bei zufällig gewichteter WRCB-Aufteilung.

4.2.5 Vergleich der Aufteilungsalgorithmen

Abbildung 4.4 zeigt die Oberfläche bei gleichmäßiger Aufteilung des Einheitsquadrates mit den vorgestellten Algorithmen. Wie man sieht, erreicht die Aufteilung mit RCB nahezu die minimale Oberfläche, während Striping wesentlich schlechtere Ergebnisse liefert. Der Übergang zu den gewichteten Algorithmen ändert nichts an diesem Bild, da die Oberfläche der Striping-Aufteilung gegenüber den Gewichten invariant ist, während sich die Oberfläche bei WRCB stellenweise sogar reduziert (siehe Abbildung 4.6).

Da der Checkerboard-Algorithmus nicht verallgemeinerbar ist, verbleibt WRCB als praktikabler Algorithmus, der das Schema mit der geringsten Oberfläche erzeugt. Die Zerlegung mit SFC liefert eine fast ebenso gute Oberfläche. Da jedoch für die Implementierung der Algorithmen nur begrenzte Zeit zur Verfügung steht, muss im Rahmen dieser Arbeit auf die experimentelle Untersuchung von SFC-Aufteilungen verzichtet werden.

Die guten Oberflächeneigenschaften des WRCB-Schemas werden mit einem Mehraufwand bei den Migrationskosten erkauft. Während bei einer Striping-Aufteilung Knoten mit benachbarten Gebieten jederzeit Last austauschen können, ohne dass dies andere Knoten berührt, ist dies bei WRCB nicht der Fall: Der WRCB-Algorithmus erzeugt einen Rekursionsbaum, an dessen Blättern sich die Rechenknoten P befinden. Bei einer Lastmigration zwischen den Knoten p_i, p_j sind alle Blätter betroffen, die unterhalb des nahesten gemeinsamen Vorfahrens von p_i und p_j liegen. Im ungünstigsten Fall sind also alle Knoten von einer Migration betroffen. Zu einem sprunghaften Anstieg des Migrationsaufwands kann es außerdem kommen, wenn ein Rechteck, welches bei D_{alt} entlang der Horizontalen geteilt wurde, wegen einer

geringfügigen Größenänderung bei D_{neu} entlang der Vertikalen geteilt wird. Die Häufigkeit derartiger Situationen im Laufe einer Simulation ist allerdings schwer vorhersehbar. Unter der Annahme, dass Lastmigration wesentlich seltener durchgeführt wird als der Austausch von Randwerten, ist der Mehraufwand bei der Migration gerechtfertigt.

Damit sich die guten Oberflächeneigenschaften von WRCB auch auf die Multiclusterumgebung übertragen lassen, müssen die rekursiven Teilungen in den obersten Rekursionsebenen entlang der Clustergrenzen erfolgen. Das heißt, es erfolgt zuerst eine gewichtete Aufteilung des Gitters unter den Clustern $\{cluster_0, \ldots, cluster_{m-1}\}$ und anschließend eine Aufteilung unter den Knoten jedes Clusters. Die Inter-Cluster-Oberfläche der so berechneten Aufteilung unter m Clustern folgt in ihrer Entwicklung dem Verlauf der gesamten Kommunikationsoberfläche für m Knoten. Die Zahl derjenigen Zellen die über langsame Clusterverbindungen versandt werden müssen wird so annähernd minimal.

Ein weiteres Argument für die Wahl von WRCB besteht darin, dass diese Aufteilung automatisch rechteckige Gebiete liefert. Dies vereinfacht die Implementierung erheblich.

4.3 Implementierung

Um die einzelnen Komponenten der Simulationsanwendung voneinander zu entkoppeln, muss die Aufteilung D des Simulationsgitters als eigenständige Datenstruktur repräsentiert werden. Diese Aufgabe wird in *MuCluDent* von der Klasse *Partition* übernommen. Diese Klasse muss die Zuordnung zwischen Rechenknoten und Koordinaten in beiden Richtungen effizient unterstützen. Zu diesem Zweck stellt *Partion* die beiden Funktionen b_1 und b_2 zur Verfügung. Die Funktion $b_1 : P \to 2^C$ ordnet jedem Knoten seine Menge von Koordinaten zu und ist als Tabelle realisiert. Da mit WRCB nur rechteckige Gebiete erzeugt werden, ist eine effiziente Darstellung durch Ursprungskoordinate, Breite und Höhe möglich. Die Funktion $b_2 : C \to P$ ordnet jeder Koordinate den passenden Knoten zu. Sie ist als Binärbaumsuche auf den bei der rekursiven Teilung erzeugten Rechtecken implementiert. Ihre Rechenzeit ist daher unabhängig von der Gittergröße und wächst logarithmisch mit der Anzahl der Knoten.

Bei der Instanziierung der Klasse *Partition* wird der WRCB-Algorithmus angewandt und die Tabelle sowie der Binärbaum angelegt. Dabei müssen wiederholt Rechtecke und Knotenmengen im richtigen Verhältnis aufgeteilt werden. Zu den Eingabedaten dieses Teilungsprozesses gehört der gemessene Systemzustand sowie eine Repräsentation der Multiclusterstruktur. Die Multiclusterstruktur wird von der Klasse *ClusterTable* bereitgestellt und speist sich momentan aus einer Konfigurationsdatei. Das genaue Zusammenspiel zwischen Systemzustand und Teilung wird in Abschnitt 5.1 behandelt.

Durch eine leichte Modifikation bei der Implementierung des Teilungsalgorithmus kann die Klasse *Partition* auch zur Berechnung und Repräsentation der ge-

wichteten Striping-Aufteilung genutzt werden. Es genügt, sich bei der Wahl der Zerlegungsrichtung auf die Horizontale oder Vertikale festzulegen, statt immer entlang der längsten Dimension zu teilen. Auf diese Weise erhält man eine Aufteilung in horizontale beziehungsweise vertikale Streifen.

Sobald eine Instanz der Klasse *Partition* initialisiert wurde, kann sie zur Steuerung des Randwerteaustausches genutzt werden. Dazu wird die Nachbarschaftsfunktion des kontinuierlichen Automaten herangezogen. Jede Nachbarschaft lässt sich in ein Quadrat mit der Kantenlänge $2r+1$ einbetten. Es genügt also, das Gebiet eines Knotens um einen äußeren Rand der Breite r zu erweitern und somit allen Zellen die benötigten Nachbarzellen zur Verfügung zu stellen. Entsprechend bildet ein r Zellen in das Gebiet hineinragender Streifen den inneren Rand. Das Rechteck, welches unter Abzug des inneren Randes vom Gesamtgebiet eines Knotens verbleibt, bildet den sogenannten *Kern*.

Zur Überlappung von Berechnung und Kommunikation werden zunächst alle Zellen des inneren Randes in den Folgezustand überführt. Die aktualisierten Zellen können sodann mit den nichtblockierenden MPI-Funktionen `ISEND` und `IRECV` übertragen werden, während der noch verbleibende Kern ebenfalls aktualisiert wird. Um die Anzahl der versandten Nachrichten zu reduzieren, werden alle Zellen des inneren Randes, welche für den selben Empfänger bestimmt sind, zu einer einzigen Nachricht gebündelt. Dazu werden auf jedem Knoten speziell angepasste MPI-Datenstrukturen erzeugt und nach jeder Lastmigration erneuert.

In den vorangegangen Abschnitten wurden Kriterien für die Wahl eines Aufteilungsalgorithmus aufgestellt und gezeigt, dass der WRCB-Algorithmus diese erfüllt. Außerdem wurde die Implementierung von WRCB durch die Klasse *Partition* beschrieben. Die anschließende Betrachtung des Lastausgleichs stützt sich auf die in diesem Kapitel getroffene Entscheidung.

5 Lastausgleich

Wie in Abschnitt 2.3 erläutert, besteht ein DLB-Algorithmus aus der Beantwortung der Fragen „Wie soll Last ausgeglichen werden?“ und „Wann soll Last ausgeglichen werden?“ Zunächst soll die Struktur des entwickelten Algorithmus anhand dieser Fragen vorgestellt werden. Die Komponenten zur Lösung der dabei anfallenden Teilprobleme werden in den folgenden Abschnitten detailliert behandelt.

Mit der Wahl des Aufteilungsalgorithmus WRCB kann die Frage des „Wie?“ angegangen werden. Ausgangspunkt des Lastausgleichs ist die Modellierung des Systemzustandes auf Basis von Laufzeitmessungen. Solch ein *Lastmodell* muss Gewichte für den Teilungsalgorithmus von WRCB zur Berechnung einer neuen Aufteilung D_{neu} liefern. Aus dem Vergleich der vorliegenden Aufteilung D_{alt} mit D_{neu} ergeben sich die anfallenden Kommunikationsvorgänge. Da zur Berechnung einer neuen Aufteilung die Gewichte aller Knoten benötigt werden, empfiehlt sich ein zentraler DLB-Algorithmus. Einem Knoten fällt die Sonderrolle zu, die Messdaten aller Knoten zu sammeln, den Systemzustand zu modellieren und das daraus berechnete D_{neu} unter den restlichen Knoten zu verteilen.

Schwieriger gestaltet sich die Beantwortung des „Wann?“: Der eingesetzte DLB-Algorithmus soll möglichst wenig an die korrekte Justierung von Parametern gebunden sein. Dies erspart die Durchführung aufwendiger Parameterstudien und ermöglicht erst den flexiblen Lastausgleich für eine Anwendung, deren Lastdynamik sich von Programmversion zu Programmversion, ja sogar von einer Probleminstanz zur nächsten, verändern kann.

Als hierfür geeigneten Algorithmus erscheint das in Abschnitt 2.3 vorgestellte Verfahren von Ka-Yeung Kwok et. al. [12]. Durch die wiederholte Schätzung der Zeitersparnisse und Kosten einer Lastmigration an den sogenannten *Entscheidungspunkten* können Situationen identifiziert werden, in denen sich der Lastausgleich lohnt, ohne dass dazu ein Schwellwertparameter gesetzt werden müsste. Als einziger Parameter verbleibt die Häufigkeit, mit der diese Überprüfung stattfindet, die sogenannte *Messperiode*. Da es jedoch nur bei Bedarf zu einer Lastmigration kommt, sollte eine niedrige Messperiode keine nachteilige Auswirkung haben. Zwischen Entscheidungspunkten und Messperiode besteht der folgende Zusammenhang mit der Zahl der Simulationsschritte des PCA:

$$\textit{Simulationsschritte} = \textit{Messperiode} \cdot \textit{Anzahl der Entscheidungspunkte} \tag{5.1}$$

Das oben beschriebene Verfahren schätzt Ersparnisse ab, die zum aktuellen Entscheidungspunkt über die folgende Messperiode hinweg erzielt werden könnten. Jedoch ist dieser Vorhersagehorizont von der Länge einer einzigen Messperiode

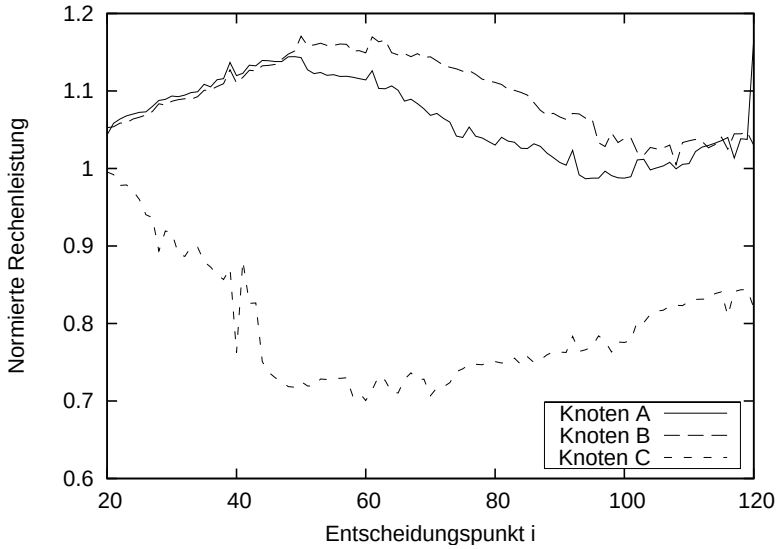


Abbildung 5.1: Normierte virtuelle Rechenleistung dreier Knoten, gemessen während eines *MuCluDent*-Simulationslaufes.

möglicherweise nicht ausreichend, um eine korrekte Migrationsentscheidung zu fällen. Wie in Abschnitt 6 gezeigt wird, kann dieser Einfluss der Messperiode jedoch zu einer erneuten Parameterabhängigkeit führen.

Dieses Problem ließe sich vermeiden, wenn eine Möglichkeit gefunden würde, den Vorhersagehorizont unabhängig von der Messperiode zu erweitern und statt der unmittelbaren die langfristigen Zeitersparnisse einer Lastmigration abzuschätzen. Eine entsprechende Heuristik wird in von Ke-Yeung Kwok et. al vorgestellt [12]. Bei dieser Heuristik, im Rahmen dieser Arbeit als *Remaining Steps Scaling (RSS)* bezeichnet, wird die unmittelbare Zeitersparnis mit der noch verbleibenden Anzahl an Simulationsschritten skaliert.

Die experimentelle Untersuchung von RSS zeigt jedoch, dass diese Skalierung bei der oszillierenden Lastdynamik von *MuCluDent* zu einer Überbewertung der Migrationsgewinne und damit zu einer Reduktion der DLB-Effizienz in Abhängigkeit vom Parameter der Messperiodenlänge führen kann. Abbildung 5.1 zeigt exemplarische Messungen schwankender Rechenlasten während eines Simulationslaufes. Eine detaillierte Erklärung dieser Messwerte findet sich in Abschnitt 5.1.1.

Im Rahmen dieser Arbeit wurde deswegen eine neue Heuristik entwickelt, welche die Parameterabhängigkeit weitgehend beseitigt. Sie beruht auf der Auswertung von Lastdaten zurückliegender Messperioden und trägt den Namen *Predicted Long-Term Savings (PLS)*. Kernstück von PLS ist die *Vorhersagefunktion.* Sie interpretiert die zurückliegenden Lastdaten mithilfe eines *Fluktuationsmodells.* Weitere Komponenten sind das *Lastmodell* zur Repräsentation des Systemzustandes und zur Abschätzung der unmittelbaren Migrationsersparnisse, sowie das *Migrationskostenmodell* zur Auswertung des Zeitaufwandes bislang durchgeführter Lastmigrationen.

5.1 Lastmodell

Informationen über die Auslastung der einzelnen Rechenknoten werden durch die Messung der für k Simulationsschritte benötigten *effektiven Rechenzeiten* t_{ij} gewonnen. t_{ij} ist die vom Knoten j in der i-ten Messperiode für die Aktualisierung seiner Zellen verbrauchte Zeit. Unter der Annahme, dass sie keinen Beitrag zum Ungleichgewicht leistet, wird die Kommunikationszeit dabei bewusst ausgeklammert.

Auf Basis dieser Messungen soll der Nutzen des Lastausgleichs quantifiziert und gegebenenfalls eine neue Lastaufteilung D_{opt} berechnet werden. Insbesondere muss die Laufzeit beliebiger Aufteilungen bei gegebenen Lastmessungen abgeschätzt werden.

Gesucht wird also ein Modell des Systemzustandes Z mit folgenden Eigenschaften

1. Z_i muss aus den Laufzeitmessungen t_{ij} (und evtl. weiteren Messungen) bestimmbar sein,
2. zu jedem Z soll eine lastbalancierte Aufteilung D_{opt} berechenbar sein,
3. zu jeder Aufteilung $D = (d_0, \cdots, d_n)$ soll die benötigte Rechenzeit für k Simulationsschritte $\mathrm{T}(Z, D)$ berechenbar sein.

Ausgehend von den in 2.3 angeführten Gründen für ungleich verteilte Last muss sowohl die Leistungsfähigkeit der Rechner als auch die wechselnde Schwierigkeit der Berechnung modelliert werden. Der Systemzustand Z muss also jedem Knoten j eine Leistungsfähigkeit P_j zuweisen. Außerdem muss für jedes Gebiet d_j der Rechenaufwand $\mathrm{complexity}(d_j)$ festgelegt werden. Der Zusammenhang dieser Größen mit der effektiven Rechenzeit des Knotens j ergibt sich dann als:

$$\mathrm{T}_j(Z, D) = \frac{\mathrm{complexity}(d_j)}{P_j} \tag{5.2}$$

Da der Randwerteaustausch schnellere Knoten zum Warten auf ihre langsameren Nachbarn zwingt, muss zur Berechnung der Gesamtlaufzeit T noch ein Maximum-Term eingefügt werden.

$$\mathrm{T}(Z, D) = \max_j T_j(Z, D) \tag{5.3}$$

Im Folgenden werden zwei Strategien zur Schätzung der Modellparameter P_j sowie zur Berechnung der Funktion complexity vorgestellt.

5.1.1 Homogenes Lastmodell

Nimmt man an, dass alle Zellen des Simulationsgitters die gleiche Rechenzeit benötigen, lässt sich die Leistungsfähigkeit der Knoten leicht ermitteln. Man setzt

complexity(d_j) proportional zur Größe von d_j und erhält:

$$P_j = \frac{|d_j|}{t_j} \tag{5.4}$$

wobei t_j die aktuell gemessene effektive Rechenzeit von p_j bezeichnet.

Da jedoch bei *MuCluDent* die Annahme der Homogenität verletzt ist, kann P_j nicht mehr als Rechenleistung des j-ten Knoten interpretiert werden. Stattdessen sind in P_j die Rechenleistung von p_j und die Schwierigkeit von d_j zu einer einzigen Maßzahl zusammengefasst. Aus diesem Grund wird P_j als *virtuelle Rechenleistung* des Knotens p_j bezeichnet. Abbildung 5.1 zeigt einen beispielhaften Verlauf der virtuellen Rechenleistungen nach einer Normierung des Mittelwertes aller P_j auf 1.

Die Laufzeit einer alternativen Aufteilung mit Gebiet d'_j kann auch auf Basis der virtuellen Rechenleistung geschätzt werden. Der dabei auftretende Schätzfehler ist umso geringer, je mehr sich die Gebiete d_j und d'_j überdecken. Laufzeitmessungen, die die Effektivität dieser Modellierung im heterogenen Fall bestätigen, finden sich in Abschnitt 6.2. Wird der so modellierte Systemzustand zur Berechnung einer neuen Aufteilung D_{neu} eingesetzt, können die P_j direkt als Gewichtsvektor verwendet werden.

5.1.2 Heterogenes Lastmodell

Das heterogene Lastmodell hat den Anspruch, lokal schwankenden Rechenbedarf explizit zu modellieren. Dabei wird versucht, Eigenschaften des physikalischen Modells auszunutzen. Wie in 2.1.2 angeführt, lässt sich die Menge aller Zellzustände grob in drei Kategorien einordnen. Die Kategorie einer Zelle ist hauptverantwortlich für den Rechenbedarf der Übergangsfunktion. Dies gibt Anlass zu folgender Modellierung: Seien l die Anzahl der Kategorien und $w = (w_1, \ldots, w_l)$ Gewichte, die dem relativen Rechenbedarf den einzelnen Kategorien entsprechen. Für einen Knoten j sei ferner $c_j = (c_{j1}, \ldots, c_{jl})$ die Anzahl der Zellen in jeder der l Kategorien zu Beginn der aktuellen Messperiode. Dann ergibt sich dessen Leistungsfähigkeit als:

$$P_j = \frac{w^T c_j}{t_j} \tag{5.5}$$

Da sich die Kategorie einer Zelle unmittelbar als Merkmal auslesen lässt, kann der Vektor c_j durch Betrachtung jeder Zelle von d_j ausgezählt werden. Zur Berechnung von w wird ein nicht-negatives lineares Regressionsverfahren herangezogen [18].

Soll nun auf Basis dieser Modellierung eine neue WRCB-Aufteilung berechnet werden, muss der Teilungsalgorithmus sowohl die P_j als auch die c_j berücksichtigen. Um ein Reckeck vertikal unter zwei Knotenmengen A, B aufzuteilen, müssen zunächst die Gewichte aller vertikalen Streifen der Breite eins ermittelt werden.

Dies wiederum erfordert den Zugriff auf die Kategorien aller Zellen des kompletten Simulationsgitters. Es ist also notwendig diese Daten auf dem zentralen DLB-Knoten zu aggregieren. Sind die Gewichte aller Streifen ermittelt, kann nach einer Teilung gesucht werden, bei der das Gewichtsverhältnis der Teilgebiete dem Verhältnis der Knotenleistungen von A und B entspricht.

Trotz verfeinerter Modellierung konnte das Heterogene Lastmodell im Praxistest (siehe Abschnitt 6.2) nicht überzeugen. Eine Analyse der Messergebnisse erbrachte folgende Gründe für das Scheitern:

1. Der Algorithmus verursacht signifikanten Rechenaufwand beim Auszählen der Zellkategorien. Sollte der durchschnittliche Rechenbedarf einer Zelle durch Änderungen am physikalischen Modell steigen, könnte dieser Mehraufwand jedoch insignifikant werden.

2. Der Algorithmus verursacht signifikanten Kommunikationsaufwand beim Aggregieren der aktuellen Kategorien aller Zellen auf dem zentralen DLB-Knoten. Sollte der durchschnittliche Speicherbedarf einer Zelle durch Modelländerungen steigen, könnte die Signifikanz dieses Mehraufwands sinken. Letzteres gilt jedoch nur bei konstanter Größe der Simulationsinstanzen, da die Anzahl der Zellen asymptotisch schneller wächst als die Kommunikationsoberfläche.

3. w kann nicht mit ausreichender Sicherheit geschätzt werden. Dies liegt zum Teil daran, dass eine gute Konditionierung des zu Grunde liegenden Gleichungssystemes nicht gewährleistet ist. Schwerer wiegt jedoch, dass beim Auszählen der Zellkategorien nur der Zustand am Anfang einer Messperiode berücksichtigt wird, die Kategorie sich jedoch innerhalb einer Messperiode ändern kann.

4. Vorhersagen über mehrere Simulationsschritte hinweg sind zu ungenau, da sich die Zellkategorien zu schnell wandeln, der Vektor c_j aber nur zum Entscheidungspunkt bekannt ist. Selbst bei korrekt bestimmtem w kann es also zu einer Fehleinschätzung des durchschnittlichen Gebietsgewichtes kommen.

Da sich die Schätzgenauigkeit des heterogenen Lastmodells gegenüber dem homogenen Modell in der Praxis nicht verbessert und darüberhinaus ein erheblicher Mehrauwand festzustellen ist, wird von der Verwendung des heterogenen Modells abgesehen.

5.2 Migrationskostenmodell

Jede Lastmigration blockiert den eigentlichen Simulationsprozess und verursacht zeitliche Kosten t. Um unnötig häufige Migrationen vermeiden zu können, sollte t

möglichst genau bestimmt werden. Dazu wird ein Modell aufgestellt und dessen Parameter aus jeweils vorhergehenden Messungen geschätzt.

Die Kosten der Migration von D_{alt} auf D_{neu} ergeben sich aus dem Zeitbedarf für die Bestimmung der zu migrierenden Zellen und der eigentlichen Kommunikation. Hinzu kommt noch die Aktualisierung von Datenstrukturen zur Steuerung des Randwerteaustausches. Das einfachste Modell schätzt all diese Kosten als konstant ein und besteht aus einem einzigen Parameter. Dieser wird als Durchschnitt der letzten m Migrationskosten bestimmt. Das Ein-Parameter Modell mit $m = 1$ findet Anwendung bei Ka-Yeung Kwok [12].

Da die Kommunikationskosten von der zu versendenen Datenmenge abhängen, liegt der Entwurf eines verfeinerten Modells mit den Parametern $t_{konstant}, t_{variabel}$ nahe. Unter der Annahme, dass alle Knoten zeitgleich kommunizieren, ist hierfür nur der Knoten p mit der größten Anzahl versandter und empfangener Zellen relevant. Dessen Kommunikationsbedarf C_p berechnet sich als:

$$\mathrm{C}_p(D_{\text{alt}}, D_{\text{neu}}) = \max |(g_j \; g'_j \cup g'_j \; g_j) \setminus (g_j \; g'_j \cap g'_j \; g_j)| \tag{5.6}$$

g_j beziehungsweise g'_j bezeichnen hierbei die Gebiete des Knotens j unter den Aufteilungen $D_{\text{alt}}, D_{\text{neu}}$. Auf diese Weise ergibt sich das Kostenmodell

$$t = t_{konstant} + t_{variabel} \cdot C_p \tag{5.7}$$

Die Parameter $t_{konstant}, t_{variabel}$ können mit einem nicht-negativen linearen Regressionsverfahren geschätzt werden.

Um die erhöhten Kommunikationskosten einer Multiclusterumgebung zu berücksichtigen, wird das Zwei-Parameter-Modell um einen Parameter c erweitert.

$$t = a + b \cdot D + c \cdot I \tag{5.8}$$

c modelliert den Einfluss der Inter-Cluster-Kommunikation I auf die Migrationszeit unter der Annahme, dass diese Kommunikation seriell erfolgt.

$$\mathrm{I}(D_{\text{alt}}, D_{\text{neu}}) = \sum |g_j \; g'_j| \tag{5.9}$$

In diesem Fall bezeichnen g_j, g'_j die zusammengefassten Gebiete aller Knoten des j-ten Clusters unter den Aufteilungen $D_{\text{alt}}, D_{\text{neu}}$. Die vereinfachende Annahme, dass Inter- und Intra-Cluster-Kommunikation nacheinander ausgeführt werden und sich folglich addieren, wird getroffen, um die Parameter weiterhin mit einem linearen Regressionsverfahren schätzen zu können.

Bei den Mehrparametermodellen müssen außerdem genügend Messdaten vorliegen, um die Parameter verlässlich schätzen zu können. Deshalb werden die Migrationskosten anfänglich gezielt unterschätzt, um für eine ausreichende Anzahl von Migrationen und Messungen zu sorgen.

5.3 Fluktuationsmodell

Die Vorhersage langfristiger Zeitersparnisse aus der Lastmigration zum aktuellen Entscheidungspunkt i_{akt} basiert auf Lastmessungen *vorhergegangener* Messperioden $i \leq i_{\text{akt}}$. Um eine geeignete Vorhersagefunktion herzuleiten, wird jedoch ein Modell für die Lastverteilung der *nachfolgenden* Messperiode $i > i_{\text{akt}}$ benötigt. Zu diesem Zwecke wird das Lastmodell aus Abschnitt 5.1.1 weiter vereinfacht und um eine zeitliche Dimension erweitert. Unter der Annahme, dass die wesentliche Dynamik schon im Wechselspiel zweier Knoten erfassbar ist, wird zunächst die Knotenzahl reduziert.

Nun betrachtet man ein Simulationsgitter, welches unter den Knoten p_0, p_1 im Verhältnis d zu $1-d, d \in]0,1[$ aufgeteilt ist. Die Arbeitslast ist ausgeglichen, wenn beide Knoten für die Berechnung eines Simulationsschrittes die gleiche Rechenzeit benötigen. Andernfalls sinkt die Arbeitseffizienz, da der jeweils schnellere Knoten auf die Übermittlung der Randwerte durch den langsameren Knoten warten muss. Die Rechenzeit eines Knotens j hängt vom Verhältnis seiner virtuellen Rechenleistung P_j zur Größe des ihm zugewiesenen Gebiets ab (siehe Gleichung 5.4). Unter der Annahme eines stetigen Zusammenhangs zwischen P_j und d existiert bei jeder möglichen Lastverteilung ein d_{opt}, welches die Last zwischen den Knoten ausgleicht. Die Stetigkeitsforderung ist bei homogener Lastverteilung erfüllt, da die virtuelle Rechenleistung in diesem Fall der tatsächlichen Rechenleistung eines Knotens entspricht und diese wiederum gar nicht von d abhängt. Im Falle heterogener Lastverteilung muss noch gefordert werden, dass kleine Änderungen an d nur kleine Änderungen am Gebiet eines Knotens verursachen. Diese Forderung ist jedoch bei den diskutierten Aufteilungsalgorithmen Striping und RCB und bei der Beschränkung auf zwei Knoten erfüllt. Eine praktische Grenze der Stetigkeitsforderung ergibt sich aus der Diskretheit des Simulationsgitters, da nur ganze Zellen zwischen den Knoten verschoben werden können.

Ziel ist es, den Verlauf der Lastschwankungen als Schwankung des Wertes d_{opt} in Abhängigkeit vom Entscheidungspunkt i darzustellen. Ausgehend von experimentellen Messungen der virtuellen Rechenleistung P_j ist es möglich, den Verlauf der P_j mit parametrisierten Sinuskurven anzunähern (siehe Abbildung 5.1). Um das Modell weiter zu vereinfachen, wird außerdem angenommen, dass die Summe der virtuellen Rechenleistungen über die gesamte Simulation hinweg konstant bleibt: $P_0 + P_1 = P$. Aus diesen Annahmen folgt sogleich, dass $\mathrm{d}_{\text{opt}}(i)$ ebenfalls als parametrisierte Sinusfunktion darstellbar ist:

$$\mathrm{d}_{\text{opt}}(i) = amp \cdot \sin(freq \cdot i + phase) + 0.5 \tag{5.10}$$

Damit d_{opt} im Intervall $]0,1[$ liegt, muss die Amplitude entsprechend eingeschränkt werden: $amp \in]0, 0.5[$. Die Frequenz *freq* und die Phase *phase* der Sinusfunktion unterliegen keinen Einschränkungen.

5.4 Vorhersagefunktion

An jedem Entscheidungspunkt i müssen die potentiellen Zeitersparnisse mit dem Aufwand der Lastmigration verglichen werden, um vorherzusagen, ob der Lastausgleich der Anwendung einen Laufzeitvorteil bringt. Solch eine Vorhersage gründet sich immer auf die *Kontinuitätshypothese:* Zukünftige Lastverteilungen ähneln den zuletzt gemessenen Lastverteilungen: $Z_{i+1} \approx Z_i$. Auf Basis dieser Hypothese wird die für die letzte Messperiode optimale Gitteraufteilung D_{opt} als neue Aufteilung $D_{\text{neu}} = D_{\text{opt}}(i)$ gewählt. Das in Abschnitt 5.1 besprochene Lastmodell gestattet zusätzlich zur Berechnung von D_{opt} die Berechnung der Laufzeit von D_{neu} bis zum nächsten Entscheidungspunkt. Als Differenz zur Laufzeit der vorliegenden Aufteilung D_{alt} ergeben sich die *unmittelbar* erwarteten Ersparnisse savings der Messperiode $i+1$.

$$\text{savings}(i+1, D_{\text{alt}}, D_{\text{neu}}) = \text{T}(Z_i, D_{\text{alt}}) - \text{T}(Z_i, D_{\text{neu}}) \tag{5.11}$$

Fallen die unmittelbaren Ersparnisse $s = \text{savings}(i+1, D_{\text{alt}}, D_{\text{neu}})$ höher aus als die vom Migrationskostenmodell aus Abschnitt 5.2 vorhergesagten Kosten t, steht einer Lastmigration nichts im Wege. Oftmals ist jedoch $s < t$ und die Entscheidung gestaltet sich schwieriger. Bliebe die vorliegende Lastsituation längerfristig stabil, so würde sich die Migration lohnen. Schlimmstenfalls könnte sich die Lastsituation jedoch ins Gegenteil verkehren, so dass die Gesamtbilanz negativ ausfallen würde. Deswegen dürfen die unmittelbaren Ersparnisse nicht beliebig weit in die Zukunft extrapoliert werden.

Um diesem bei der RSS-Heuristik zu beobachtenden Problem zu begegnen, soll eine Vorhersagefunktion pred entwickelt werden, die den langfristigen Nutzen einer Lastmigration zum Entscheidungspunkt i_{akt} aus der Analyse vergangener Lastverteilungen $Z_i, i < i_{\text{akt}}$ abschätzt.

Das Fluktuationsmodell d_{opt} (siehe Gleichung 5.10) gestattet die Berechnung der savings-Funktion für in der Zukunft liegende Messperioden. Damit ist es möglich, den idealen Verlauf der Vorhersagefunktion idealPred zu berechnen:

$$\text{idealPred}(i_{\text{akt}}) = \sum_{i=i_{\text{akt}}+1}^{e} \text{savings}(i, D_{\text{alt}}, D_{\text{neu}}) \tag{5.12}$$

wobei e der maximimale Index ist, für den alle Summanden positiv sind.

Die Berechnung von idealPred ist nur unter Kenntnis der Modellparameter von d_{opt} möglich. In der Praxis ist jedoch der Verlauf von d_{opt} nicht vorhersagbar, da weder die Parameter bekannt sind, noch sichergestellt ist, dass sich die Last tatsächlich sinusförmig entwickelt. Die Vorhersagefunktion pred stützt sich deswegen auf die effektiv berechenbaren Werte der savings-Funktion für zurückliegende Messperioden $i \leq i_{\text{akt}}$. Dem zu Grunde liegt eine Erweiterung der Kontinuitätshypothese: Ein Trend positiver, anwachsender Ersparnisse setzt sich so weit in die

Zukunft fort, wie er sich in die Vergangenheit erstreckt.

$$\text{pred}(i_{\text{akt}}) = \sum_{i=e}^{i_{\text{akt}}} \text{savings}(i, D_{\text{alt}}, D_{\text{neu}}) \tag{5.13}$$

wobei $e < i_{\text{akt}}$ der minimale Index ist, für den folgende Bedingungen erfüllt sind:

1. Alle Summanden sind positiv.
2. Die Summanden bilden eine monoton aufsteigende Folge.

Setzt man nun in das Fluktuationsmodell d_{opt} Beispielparameter ein, so kann man den Funktionsverlauf von pred und idealPred errechnen und die Vorhersagequalität von pred bewerten. Die stark kondensierte Darstellung des Systemzustandes Z durch einen einzigen Wert d_{opt} erfordert jedoch eine modifizierte Berechnung der savings-Funktion. Anstelle absoluter Ersparnisse kann nur mit relativen Ersparnissen, gemessen an der auf 1 normierten Optimalaufzeit, gearbeitet werden. Da aber gemäß unserer Modellannahme die Summe der virtuellen Rechenleistungen P konstant ist, bleibt auch die Optimallaufzeit konstant, und relative Ersparnisse können genau so addiert und verglichen werden wie absolute Ersparnisse. Die Funktionsargumente $D_{\text{alt}}, D_{\text{neu}}$ werden gemäß der Logik des Fluktuationsmodells ebenfalls als einfache Größenverhältnisse $d_{\text{alt}}, d_{\text{neu}}$ interpretiert.

$$\text{savings}(i, d_{\text{alt}}, d_{\text{neu}}) = \text{T}_{\text{rel}}(\text{d}_{\text{opt}}(i), d_{\text{alt}}) - \text{T}_{\text{rel}}(\text{d}_{\text{opt}}(i), d_{\text{neu}}) \tag{5.14}$$

$$\text{T}_{\text{rel}}(\text{d}_{\text{opt}}(i), d) = \max\left\{ \frac{d}{\text{d}_{\text{opt}}(i)}, \frac{1-d}{1-\text{d}_{\text{opt}}(i)} \right\} \tag{5.15}$$

Abbildung 5.2 zeigt im oberen Teil den Funktionsgraphen des Fluktuationsmodells d_{opt} sowie die Aufteilungen d_{alt} und d_{neu}. Zum Entscheidungspunkt i_{akt} ist d_{neu} die optimale Aufteilung, denn $\text{d}_{\text{opt}}(i_{\text{akt}}) = d_{\text{neu}}$. Der untere Teil von Abbildung 5.2 zeigt die relativen Ersparnisse savings, die sich ergäben, wenn in der Messperiode i die Aufteilung d_{neu} an Stelle von d_{alt} vorläge.

In Abbildung 5.3 wird der Verlauf der Funktionen idealPred, pred und savings dargestellt. Obwohl pred die wahren Langzeitersparnisse wesentlich unterschätzt, stellt diese Funktion im Vergleich zu den unmittelbaren Ersparnissen savings eine viel bessere Näherung an idealPred dar. Eine geringe Überschätzung der Ersparnisse geschieht an den Scheitelpunkten der Sinusschwingung von d_{opt}. Dies sind genau die Stellen, an denen die erweiterte Kontinuitätshypothese anhaltender Trends verletzt wird.

Die Betrachtung eines einzigen Parameterbeispiels kann noch kein umfassendes Bild der Vorhersagegenauigkeit von pred liefern. Aus diesem Grunde muss betrachtet werden, wie sich das Verhältnis von idealPred zu pred bei Parameteränderung verhält. Die Funktionswerte von pred sowie idealPred ergeben sich durch Summation konsekutiver Werte der savings-Funktion. Somit ist der Wert beider Funktionen

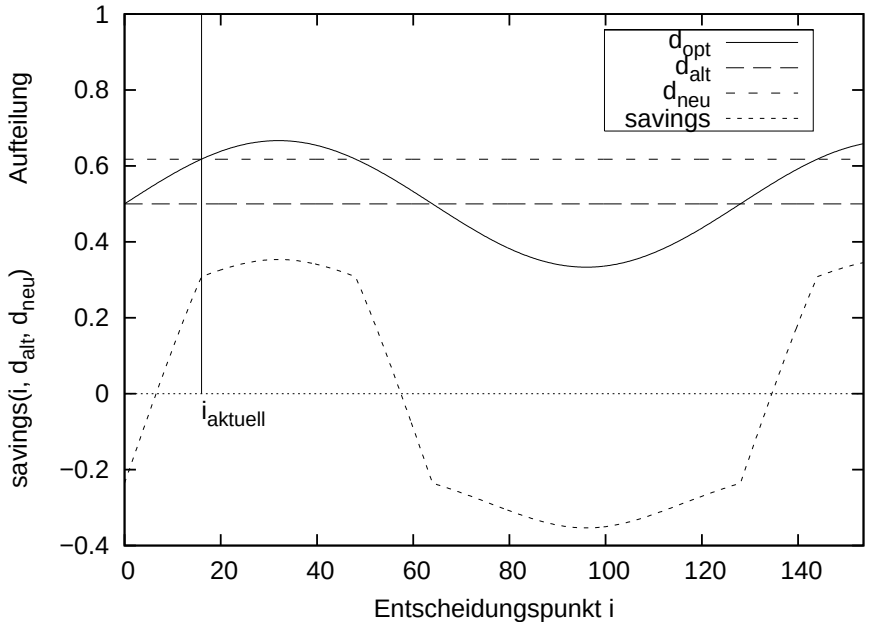


Abbildung 5.2: Relative Zeitersparnis von $d_{\text{neu}} = \mathrm{d}_{\text{opt}}(i_{\text{akt}})$ gegenüber d_{alt} basierend auf dem Fluktuationsmodell d_{opt}.

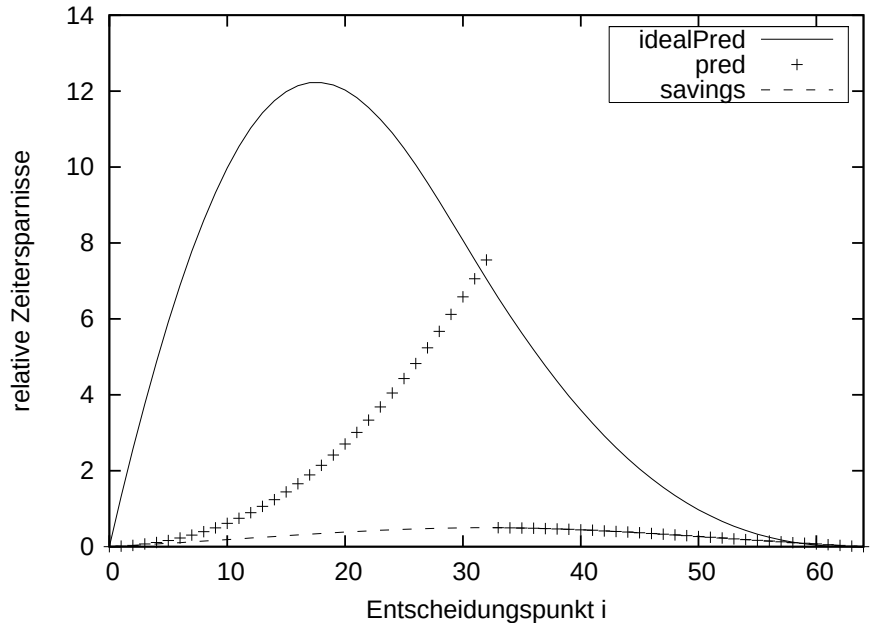


Abbildung 5.3: idealPred im Vergleich zu pred und savings (Parameter: $phase = 0, freq = \frac{\pi}{64}, amp = \frac{1}{6}$).

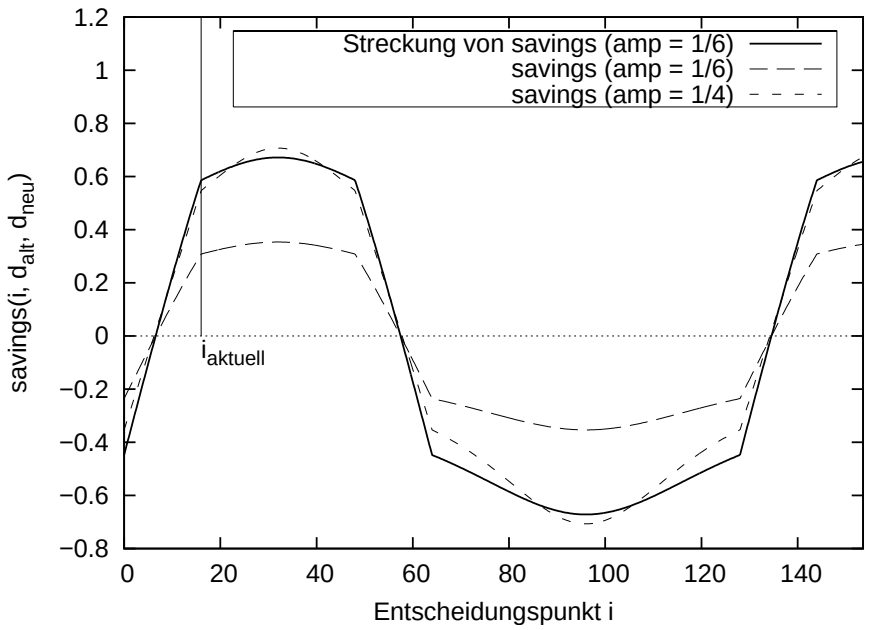


Abbildung 5.4: Einfluss des Parameters *amp* auf die savings-Funktion.

proportional zu einer Fläche unterhalb der savings-Kurve. Wie aus der Geometrie bekannt, ist das Flächenverhältnis invariant unter der Gruppe affiner Transformationen. Zum Beweis der allgemeinen Anwendbarkeit von pred genügt es also, die Parameteränderungen von d_{opt} auf affine Transformationen der savings-Funktion zurückzuführen.

phase Eine Änderung dieses Parameters verursacht eine Verschiebung von savings entlang der Zeitachse.

freq Eine Änderung dieses Parameters entspricht einer Parallelstreckung von d_{opt}. Diese Streckung überträgt sich unmittelbar auf die savings-Funktion.

amp Eine Änderung dieses Parameters entspricht einer Parallelstreckung von d_{opt}. Dies führt näherungsweise zu einer Streckung der savings-Funktion (siehe Abbildung 5.4). Es genügt, den Kurvenverlauf im positiven Bereich zu betrachten, da nur über diesen summiert wird.

Ein Vergleich der Vorhersagegenauigkeit von RSS mit idealPred gestaltet sich problematisch, da der Skalierungsfaktor dieser Heuristik von der Anzahl noch verbleibender Entscheidungspunkte abhängt. In Abhängigkeit vom Anwendungsfall kann dieser also beliebig hoch werden.

Nachdem nun alle Komponenten des DLB-Algorithmus vorgestellt wurden, können die Heuristiken empirisch untersucht werden. Die Ergebnisse und Analyse der mit PLS und RSS vorgenommenen Laufzeitmessungen finden sich in Abschnitt 6.3.

6 Auswertung

Die komponentenbasierte Architektur von *MuCluDent* gestattet es, Entwurfsentscheidungen hinauszuzögern und jede einzelne Komponente unabhängig von den anderen zu optimieren. Doch wie mit jeder Freiheit ist auch mit dieser die Verantwortung verbunden, jede Entscheidung sorgfältig abzuwägen. Kernthemen dieser Arbeit sind die Wahl eines geeigneten Aufteilungsalgorithmus und die Entwicklung eines parameterunabhängigen DLB-Algorithmus. Letzterer gliedert sich in weitere Komponenten, von denen das Lastmodell und die Vorhersagefunktion besonders intensiv untersucht wurden. Die bei der Implementierung dieser Komponenten getroffenen Entscheidungen sollen in den folgenden Abschnitten experimentell bewertet werden. Dabei wird versucht, den Einfluss der jeweils untersuchten Komponente so weit wie möglich isoliert von den restlichen Entscheidungen zu betrachten. Alle Messungen wurden auf dem in Anhang B beschriebenen Multi-Domain-Cluster durchgeführt.

6.1 Messungen zum Aufteilungsschema

Die in Abschnitt 4 getroffene Entscheidung für den Aufteilungsalgorithmus WRCB erfolgte allein auf Grund theoretischer Überlegungen bezüglich des erzeugten Aufteilungsschemas. Um die praktische Auswirkung dieser Wahl zu überprüfen, wird die Simulation unter Einsatz des WRCB- sowie des Striping-Algorithmus durchgeführt und die Laufzeiten gemessen. Abbildung 6.1 zeigt den auf Cluster A erzielten Speedup bei einer Probleminstanz mit $2^9 \times 2^9$ Zellen und 2^{10} Schritten. Es zeigt sich, dass der theoretisch vorhergesagte Vorteil einer geringeren Oberfläche und damit geringeren Kommunikationsbedarfs zu einem messbaren Zeitvorteil der WRCB-Aufteilung gegenüber Striping führt. Wie durch den Kurvenverlauf in Abbildung 4.4 zu erwarten, verstärkt sich dieser Vorteil bei wachsender Knotenzahl. Aus den Messergebnissen ist außerdem die Effektivität der Parallelisierung ersichtlich: Die Simulationsanwendung erfährt bei der Berechnung auf Cluster A eine Beschleunigung um den Faktor $6,7$.

Damit ist jedoch noch nicht geklärt, ob die Simulation auch vom Einsatz eines Multiclusters profitieren kann. Zu diesem Zweck wurden Messungen mit unterschiedlichen Clusterkonfigurationen, Aufteilungsalgorithmen und Instanzgrößen durchgeführt. In Tabelle 6.1 sind die für 2^{10} Simulationsschritte gemessenen Laufzeiten in Sekunden angegeben.

Wie nach der Speedup-Messung zu erwarten, werden bei Verwendung der WRCB-Aufteilung durchweg bessere Laufzeiten erzielt als bei Striping. Dies führt jedoch

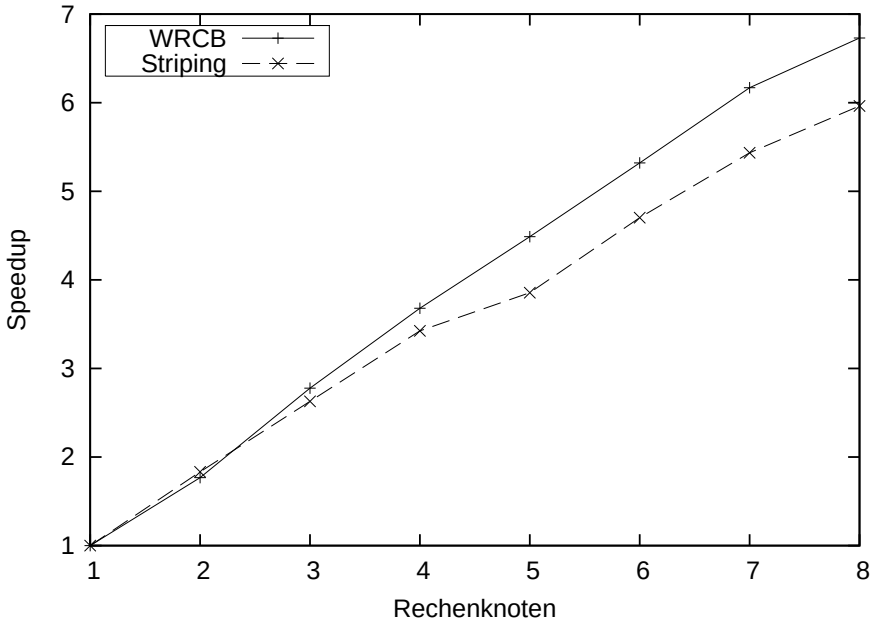


Abbildung 6.1: Speedup in Abhängigkeit von der Knotenzahl und dem Aufteilungsschema.

	Aufteilungsschema	**A**	**B**	**A + B**
$\mathbf{2^{18}}$ **Zellen**	WRCB	64	80	42
	Striping	71	106	72
$\mathbf{2^{20}}$ **Zellen**	WRCB	243	270	147
	Striping	253	347	220

Tabelle 6.1: Laufzeit der Simulation bei verschiedenen Aufteilungsalgorithmen, Problemgrößen und Hardwarekombinationen (Angaben in Sekunden).

zu interessanten Ergebnissen beim Vergleich unterschiedlicher Clusterkonfigurationen: Bei einer Problemgröße von 2^{18} Zellen kann durch den Verbund der Cluster *A* und *B* ein Zeitgewinn gegenüber der Berechnung auf Einzelclustern erreicht werden. Dies gilt jedoch nur bei der Verwendung von WRCB. Erst durch die Erhöhung der Problemgröße um den Faktor 4 ist eine derartiger Gewinn auch beim Einsatz von Striping zu verzeichnen.

Zusammenfassend kann also gesagt werden, dass durch die Wahl von WRCB gegenüber Striping die Laufzeit allgemein verbessert, und außerdem das Einsatzgebiet erweitert wird. Bei Verwendung des kommunikationssparenderen Aufteilungsschemas erhöht sich die Menge der Simulationsprobleme, die vom Einsatz eines Multiclusters profitieren.

6.2 Messungen zum Lastmodell

Das in Abschnitt 5.1.1 vorgestellte Homogene Lastmodell beruht auf einer starken Vereinfachung der tatsächlichen Lastverteilung. Bei räumlich stark schwankender Rechenlast kann es theoretisch vorkommen, dass die als vorteilhaft berechneten Neuaufteilungen das Ungleichgewicht sogar noch verstärken. Um diesem Problem zu begegnen, wurde ein alternatives Modell, das Heterogene Lastmodell entwickelt. Dieses wiederum hat den Nachteil hohen Mehraufwands bezüglich Rechenlast und Kommunikation. Da die praktische Konsequenz dieser Vor- und Nachteile vom typischen Anwendungsverhalten und der eingesetzten Infrastruktur abhängen, muss die Entscheidung für eines dieser Modelle letztlich auf einer experimentellen Messung beruhen. Um eine Vergleichsgröße für die Effektivität des Lastausgleichs zu erhalten, wurde eine weiteres Lastmodell implementiert. Bei Verwendung des *Statischen Lastmodells* wird das Gitter gleichmäßig unter den Knoten aufgeteilt und es kommt zu keinem weiteren Lastausgleich.

In Tabelle 6.2 sind die beim Einsatz der verschiedenen Lastmodelle gemessenen Laufzeiten angegeben. Die Simulation von 2^{18} Zellen über 2^{10} Schritte wurde auf den 8 Knoten des Clusters *A* durchgeführt. Die Berechnung von Gitteraufteilungen geschah mit dem WRCB-Algorithmus. Zur Aufschlüsselung der anteiligen Zeiten einzelner Programmteile wurden die entsprechenden Codeabschnitte mit Aufrufen an einen hochauflösenden Zeitgeber instrumentiert.

Eine Betrachtung der Gesamtlaufzeit bestätigt die Effektivität des Lastausgleichs.

	Statisch	**Heterogen**	**Homogen**
Mit Lastausgleich verbrachte Zeit	0	29	8
Wartezeit durch Randwerteaustausch	62	14	11
Gesamtlaufzeit in Sekunden	320	302	277

Tabelle 6.2: Laufzeit der Simulation (in Sekunden) unter Einsatz verschiedener Lastmodelle.

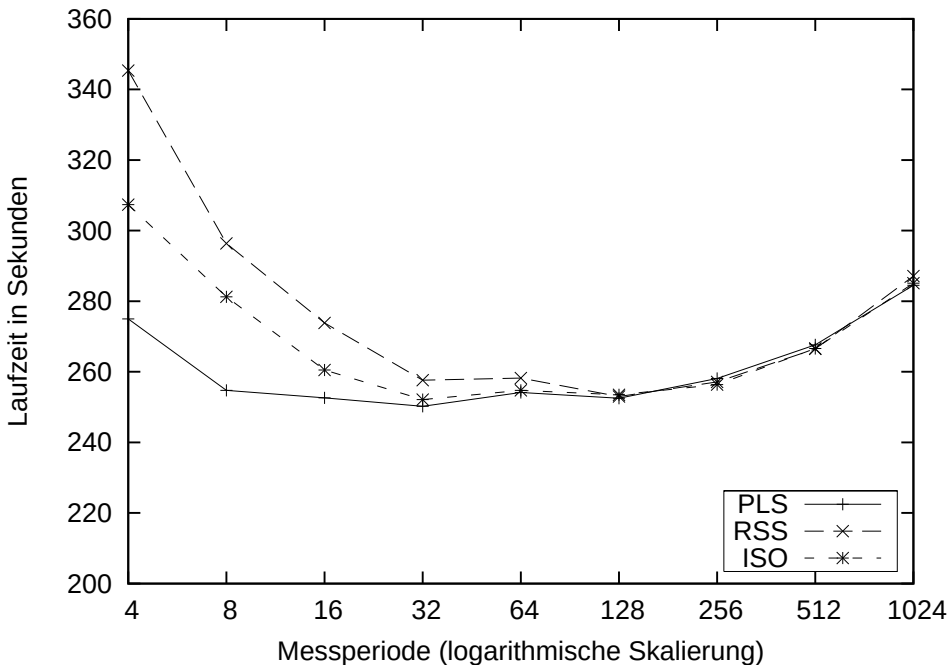


Abbildung 6.2: Laufzeit in Abhängigkeit von Vorhersage-Heuristik und Messperiode k.

Bei Verwendung des Homogenen Lastmodells kommt es zu einer 13-prozentigen Laufzeitverbesserung gegenüber der Unterlassung jeglichen Lastausgleichs. Auch beim Einsatz des Heterogenen Lastmodells kommt es zu einer Reduktion der durch Randwerteaustausch bedingten Wartezeit. Dieser Gewinn wird jedoch durch den Mehraufwand des komplexeren Modells verringert.

6.3 Messungen zur Vorhersage-Heuristik

Um die Nützlichkeit von PLS zu bewerten, wurden zwei Referenzheuristiken implementiert:

Remaining Steps Scaling (RSS) Unmittelbare Ersparnisse werden mit der noch verbleibenden Anzahl von Entscheidungspunkten multipliziert und dann mit den Migrationskosten verglichen [12].

Immediate Savings Only (ISO) Unmittelbare Ersparnisse werden direkt mit den Migrationskosten verglichen. Diese ist die einfachste Implementierung einer Kosten/Nutzen Abschätzung da keinerlei Betrachtung der Lastdynamik stattfindet.

Abbildung 6.2 zeigt die Gesamtlaufzeiten bei der Berechnung einer beispielhaften Probleminstanz (2^{18} Zellen, 2^{12} Simulationsschritte) auf 9 Rechenknoten. Bei allen eingesetzten Heuristiken variiert die Laufzeit in Abhängigkeit von der Messperiodenlänge k, erreicht jedoch in etwa denselben Minimalwert.

Wie kommt es zu diesem Kurvenverlauf? Wie in Abschnitt 5.1 beschrieben, ist die Anzahl der Entscheidungspunkte indirekt proportional zur Länge der Messperiode. Wird die Messperiode zu lang gewählt, schränkt sich daher die Reaktionsgeschwindigkeit des DLB-Algorithmus ein, da nur an jedem Entscheidungspunkt überhaupt Last migriert werden kann.

Das schlechte Abschneiden der Heuristiken RSS und ISO bei niedrigen Periodenlängen kann unterschiedlichen Ursachen zugeordnet werden. Wie in Abschnitt 5.4 dargelegt, neigt RSS dazu, die Ersparnisse im Fall oszillierender Lasten zu überschätzen. Eine erhöhte Anzahl von Entscheidungspunkten bietet daher vermehrt Gelegenheiten zu unproduktivem Lastausgleich. Im Vergleich dazu resultieren die erhöhten Laufzeiten von ISO aus dem gegenteiligen Grund: Die erwarteten, unmittelbaren Ersparnisse fallen bei kürzeren Messperioden zunehmend geringer aus. Um so seltener kommt es vor, dass eine Lastmigration allein auf Basis der unmittelbaren Ersparnisse zu rechtfertigen ist. Wird die Lastmigration jedoch zu selten ausgelöst, kommt es zu Zeitverlusten durch geringere Rechnerauslastung. Selbst beim Einsatz der PLS-Heuristik darf die Messperiode nicht beliebig klein gewählt werden, da der Rechenbedarf des Entscheidungsalgorithmus unmittelbar von der Anzahl der Entscheidungspunkte abhängt.

In der Praxis muss die Periodenlänge in Unkenntnis des genauen Kurvenverlaufs gewählt werden. Für die Wahl dieses Parameters existiert keine triviale Strategie, da sowohl zu hohe als auch zu niedrige Messperioden die Effizienz senken. Der entscheidene Vorteil von PLS gegenüber den Heuristiken RSS und ISO ergibt sich daher aus der geringeren Anfälligkeit gegenüber unpassender Parameterwahl. Beim Einsatz der letztgenannten Heuristiken muss die optimale Messperiodenlänge mit einer Parameterstudie festgestellt werden. Wohingegen es bei der Verwendung von PLS möglich ist, diesen Parameter aus einem breiten Intervall zu wählen und dabei eine nahezu optimale Laufzeit zu erzielen. Dies reduziert den Kalibrieraufwand des DLB-Algorithmus.

Die in den vorangegangen Abschnitten präsentierten Messergebnisse haben die Effektivität der Parallelisierung bewiesen und die Entwurfsentscheidungen im Rahmen der betrachteten Alternativen gerechtfertigt.

7 Zusammenfassung und Ausblick

Gegenstand der vorliegenden Arbeit ist die Entwicklung eines dynamischen Lastausgleichsalgorithmus für die parallele Dendritensimulation *MuCluDent*. Insbesondere wurde nach einem Algorithmus gesucht, der für den Einsatz in einer Multiclusterumgebung geignet ist. Eine weitere Anforderung war die Robustheit gegenüber Änderungen am physikalischen Modell der Simulation.

Um diese Ziele zu erreichen, wurde ein weitgehend parameterfreier DLB-Algorithmus entwickelt, der sich durch die fortwährende Analyse von Leistungsmessungen automatisch dem Anwendungsverhalten anpasst. Wo auf Parameter nicht ganz verzichtet werden konnte, wurde eine Heuristik entwickelt, die deren Einfluss reduziert und so die Robustheit wahrt. Ein wichtiger Aspekt war die Wahl eines geeigneten Aufteilungsalgorithmus zur Bestimmung der erforderlichen Lastmigration. Der serielle Anteil des Algorithmus und damit seine parallele Effizienz, hängen von der Oberfläche der Gitteraufteilung ab. Durch einen zentralen Ansatz, der bei jeder Migration die gesamte Aufteilung berücksichtigt, wird eine durchgehend niedrige Oberfläche sichergestellt.

Der so konstruierte Algorithmus kann ohne die Einstellung anwendungsspezifischer Parameter verwendet werden und zeichnet sich durch eine gute Skalierbarkeit aus. Bei Experimenten an einem Multicluster konnte sowohl die statische Heterogenität der Infrastruktur als auch das dynamische Ungleichgewicht der Anwendung erfolgreich ausgeglichen und eine beachtliche Geschwindigkeitssteigerung erzielt werden.

Die entwickelten Algorithmen sind allgemein genug, um in einem weiteren Kontext zur Anwendung zu kommen. Durch die Entkopplung von anwendungspezifischen Lasteigenschaften ist ein Einsatz bei beliebigen als kontinuierlicher Automat modellierbaren Anwendungen möglich. Der zur Aufteilung verwendete WRCB-Algorithmus ist problemlos auf höhere Gitterdimensionen verallgemeinerbar.

Im Laufe dieser Arbeit tauchten zahlreiche Fragestellungen auf, deren weitere Erforschung lohnend erscheint:

- Wie können die Schwächen der betrachteten Lastmodelle ausgeglichen werden? Das homogene und das heterogene Lastmodell stellen Extrempunkte einer Skala von Lastmodellen dar, insofern sie die lokal schwankende Last in einem einzigen Wert pro Knoten zusammenfassen bzw. sie mit der Auflösungsgenauigkeit einer einzigen Zelle modellieren. Vielleicht lässt sich bezüglich dieser Genauigkeit ein besserer Kompromiss finden.

- Wie sinnvoll ist das Migrationskostenmodell mit drei Parametern? Eine genaue Analyse setzt eine Vielzahl von MC-Umgebungen voraus. Eventuell ist

es vorteilhaft, zu einem theoretisch fundierteren Modell, beispielsweise *LogP*, überzugehen [19].

- Wie lässt sich die Genauigkeit der Vorhersagefunktion verbessern? Gibt es eventuell ein besseres Fluktuationsmodell? Denkbar wäre auch der Versuch, direkt die Parameter des Fluktuationsmodells zu schätzen.

- Bringt es Vorteile, an jedem Entscheidungspunkt verschieden Neuaufteilungen zu betrachten? Denkbar wäre die Berechnung einer Aufteilung, welche einen Kompromiss zwischen der (geschätzten) optimalen sowie der aktuellen Aufteilung darstellt. Eine derartige Kompromiss-Aufteilung brächte vielleicht weniger Zeitersparnisse, wäre dafür jedoch auch mit geringeren Migrationskosten verbunden.

- Welche Vor- und Nachteile sind mit der Verwendung von SFC-Aufteilungen verbunden? Diese haben zwar ein etwas höhere Oberfläche, gestatten dafür jedoch die Verwendung lokaler DLB-Strategien.

Mit Beantwortung obiger Fragen verbindet sich das Potential, den in dieser Arbeit vorgestellten DLB-Algorithmus zu optimieren und das Verständnis für selbstkalibrierende Algorithmen zu erweitern.

Literaturverzeichnis

[1] John v. Neumann. *The Theory of Self-Reproducing Automata*. The University of Illinois Press, 1966.

[2] Niloy Ganguly, Biplab K. Sikdar, Andreas Deutsch, Geoffrey Canright, and P. Pal Chaudhuri. A survey on cellular automata. Technical report, Centre for High Performance Computing, Dresden University of Technology, 2003.

[3] P.M.A. Sloot and A.G. Hoekstra. Cellular automata as a mesoscopic approach to model and simulate complex systems. volume 2074 (II) of *Lecture Notes in Computer Science*, pages 518–527, Heidelberg, May 2001. Springer Verlag.

[4] M. Mazzariol, B. Gennart, and R. Hersch. Dynamic load balancing of parallel cellular automata. volume 4118 of *SPIE Conference on Parallel and Distributed Methods for Image Processing*, 2000.

[5] Brinch Hansen. Parallel cellular automata. http://brinch-hansen.net/papers/1993c.pdf, 1993.

[6] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl. Questions and answers about bsp. Technical Report PRG-TR-15-96, Computing Laboratory, Oxford University, 1996. Also in *Journal of Scientific Programming*, V.6 N.3, 1997.

[7] Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingil. *Patterns for Parallel Programming*. Addison Wesley Professional, 2004.

[8] Michael J. Quinn and Philip J. Hatcher. On the utility of communication-computation overlap in data-parallel programs. *Journal of Parallel and Distributed Computing*, 33(2):197–204, March 1996.

[9] Adrian Knoth. IPv6 message passing mit Open MPI. Diplomarbeit, Friedrich Schiller Universität Jena, 2007.

[10] A. Plastino, C. Ribeiro, and N. Rodriguez. Load balancing algorithms for SPMD applications. CiteSeer [http://cs1.ist.psu.edu/cgi-bin/oai.cgi] (United States), 1999.

[11] Arjen Schoneveld, Martin Lees, Erwan Karyadi, and Peter M. A. Sloot. Dynamic load balancing in parallel finite element simulations. In P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, editors, *High-Performance Computing*

and Networking. Proc 7th HPNC Conference, Amsterdam, volume 1593 of *Lecture Notes in Computer Science*, pages 409–419, Berlin/Heidelberg, 1999. Springer.

[12] Ka-Yeung Kwok, Fu-Man Lam, Mounir Hamdi, and Yi Pan. *Application-Specific Load Balancing on Heterogeneous Systems*, volume 2. Prentice Hall, New Jersey, 1999.

[13] MPI-2: Extensions to the message-passing interface. The MPI Forum, July 1997.

[14] J. Hungershöfer and J.-M. Wierum. On the quality of partitions based on space-filling curves. volume 2331 of *Lecture Notes in Computer Science*, 2002.

[15] Levi Valgaerts. Space-Filling Curves: An Introduction. Technical University Munich, 2005.

[16] Mathias Goldau. Discrete space filling curves as domain decomposition for communication sensitive load balancing on multiclusters considering high performance applications such as cellular automata. Diplomarbeit, Friedrich Schiller Universität Jena, 2006.

[17] S.H. Berger, M.J. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. In *Computers, IEEE Transactions on*, volume C-36, pages 570–580, 1987.

[18] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. PrenticeHall, Englewood Cliffs, NJ, USA, 1974.

[19] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.

A Konfigurationsdatei

Beispiel einer Konfigurationsdatei für *MuCluDent*.

```
# physikalische Konstanten
stepDuration = 1.0e-10;
defaultTemperatureLiquid = 733.452;
defaultTemperatureSolid = 933.452;
cellDimX = 3.0e-7;  #\
cellDimY = 3.0e-7;  # > cell dimensions
cellDimZ = 3.0e-7;  #/
conductivity = 210.0;
meltingEnthalpy = 6.0e+8;
specificHeat = 2.58e+6;
gibbsThompson = 1.9e-7;

# Modellspezifische Parameter
maximumSteps = 1024;     # Anzahl der Simulationsschritte
diagonalPhaseBorderWeight = 0.69;
anisotropyTemperatureCoefficient = 0.01;
anisotropySpeedCoefficient = 0.28;
edgeHeatFlow = 0.0;     # Temperaturfluss am Gitterrand

# Startkonfiguration des Gitters
dimensions = [512, 512];
seeds {
    seed0 {
         coord = [100, 100];
         state = "SOLID";
         angle = 0.5;
    }
}
```

B Messinfrastruktur

Alle Messungen wurde auf den folgenden Clustern durchgeführt:

Cluster A besteht aus 8 Rechenknoten die durch Gigabit-Ethernet verbunden sind. Die Knoten haben identische Leistungsdaten: 3.0 GHz Intel Pentium 4 Prozessor, 512 KB L2 cache, 2 GB RAM.

Cluster B besteht aus 18 Rechenknoten die durch 100MBit-Ethernet verbunden sind. Die Knoten haben identische Leistungsdaten: 1.4 Ghz AMD Athlon, 256 KB L2 cache, 512 MB RAM.

Cluster A und B sind durch Gigabit-Ethernet verbunden.

C CD-Inhalt

Pfad	Beschreibung
Struktur von *MuCluDent*	
`doc`	Dokumentation von *MuCluDent*
`lib`	Externe Bibliotheken
`src/engine`	Physikalisches Modell
`src/gui`	Visualisierungsprogramm für Simulationsdaten
`src/io`	Ausgabe der Simulationsdaten
`src/lib`	Interne Bibliotheken
`src/loadbalancer`	Experimenteller Lastausgleichsalgorithmus
`src/mpi`	Abstraktionsschicht für das Message Passing Interface
`src/mucludent.cc`	Hauptprogramm
`src/parallelization`	Komponenten zur Parallelisierung
`src/test`	Allgemeine Funktionstests
Komponenten, die im Rahmen dieser Arbeit entwickelt wurden (Pfade relativ zu `src/parallelization/partition/`)	
`../partitioningsimulator.h`	Simulator
`clustertable.h`	Verwaltung der MC-Struktur
`loadmodel.h`	Abstraktes Lastmodell und Vorhersagefunktion
`homogeneous_loadmodel.h`	Homogenes Lastmodell
`state_based_loadmodel.h`	Heterogenes Lastmodell
`static_loadmodel.h`	Statisches Referenz-Lastmodell
`communicationmodel.h`	Migrationskostenmodell
`partition.h`	Abstrakte Aufteilung
`partition_striping.h`	*Striping*-Aufteilung
`partition_recursive_bisection.h`	*WRCB*-Aufteilung
`splitter.h`	*RCB*-Teilungsalgorithmus
`modelsplitter.h`	*WRCB*-Teilungsalgorithmus

Hilfskomponenten und Funktionstests (Pfade relativ zu `src/parallelization/`)	
`partition/nodes.h`	Verwaltung von Rechenknoten
`partition/treenode.h`	Element eines RCB-Baumes
`partition/test`	Funktionstests
`test`	Funktionstests

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Jena, 29.05.2007 Jakob Erdmann